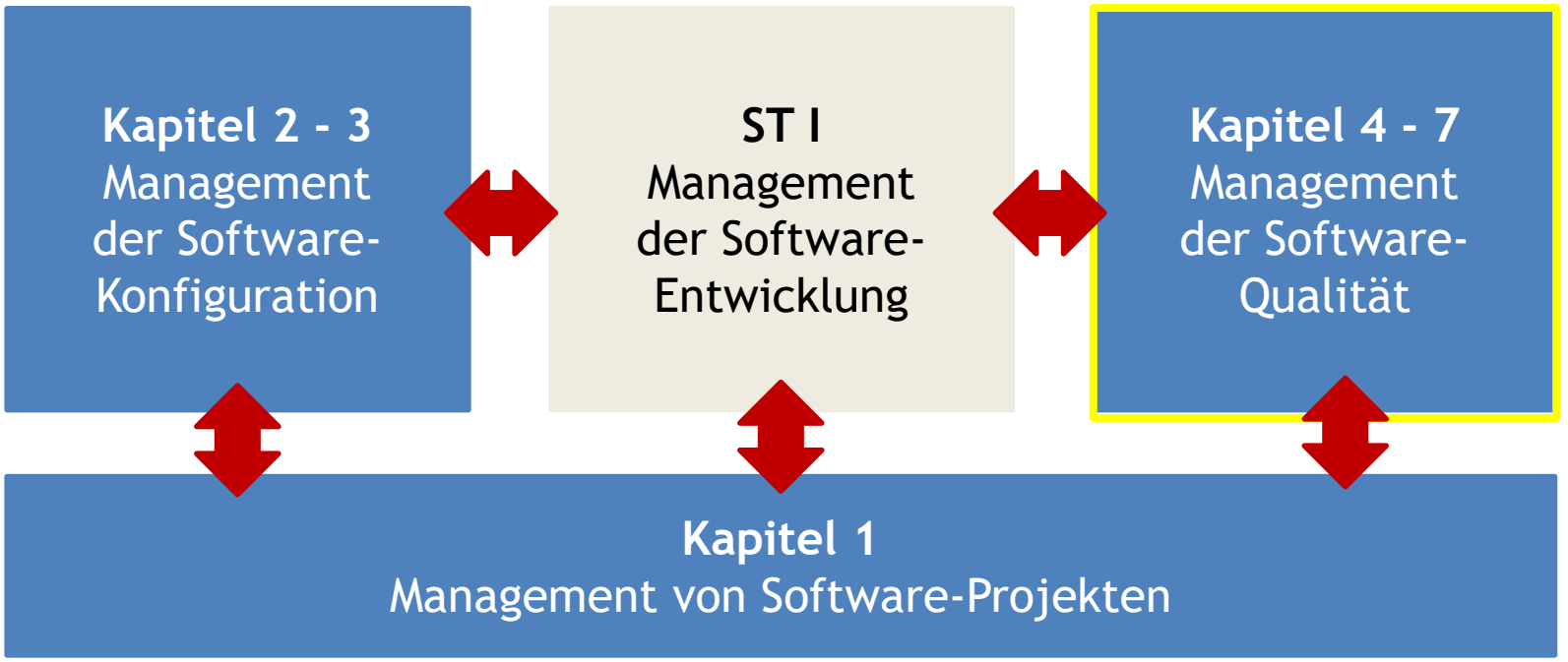
UNIVERSITÄT SIEGEN

Vorlesung Softwaretechnik II

Management der Software-Qualität: Formale Analysen

Aufbau der Vorlesung



UNIVERSITÄT SIEGEN

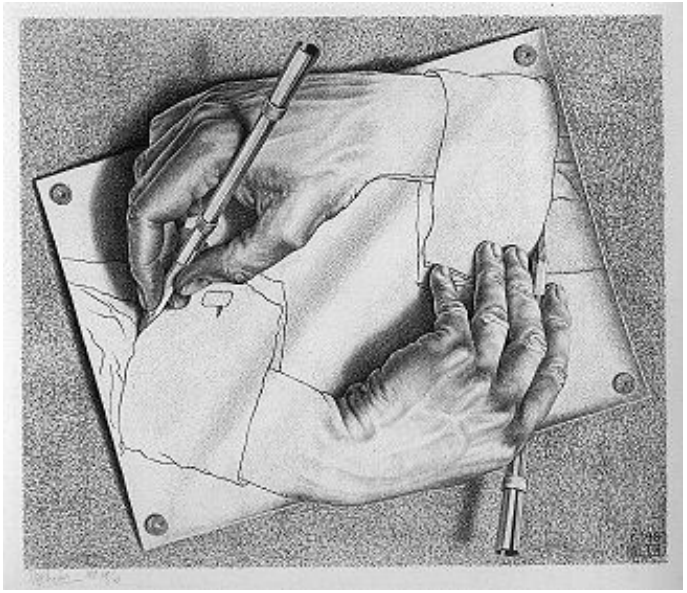
Inhalt

- Grundlagen
- Einführung in CCS
- Formale Syntax und Semantik von CCS und LTS
- Kongruenzregeln für CCS
- Äquivalenzbegriffe für LTS

UNIVERSITÄT SIEGEN

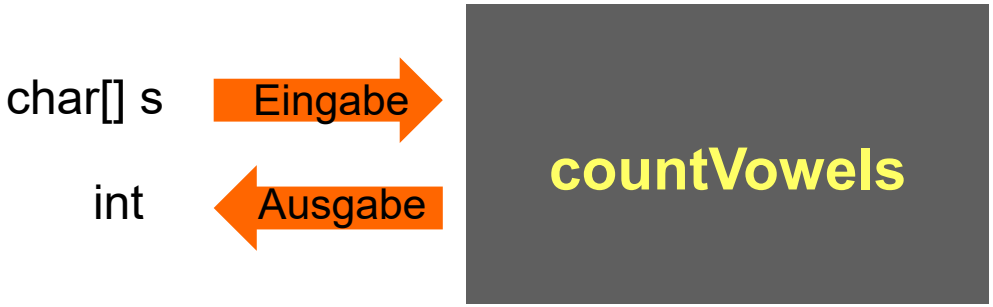
Grundlagen

Reaktive Systeme und Prozesstheorie



„Drawing Hands“, M. C. Escher, 1948

Terminierende Systeme



Bisherige Annahme: Das Software-(Teil-)System erhält eine Eingabe, führt für diese Eingabe einen mehr oder weniger komplizierten Berechnungsalgorithmus aus, **terminiert** und liefert das Berechnungsergebnis als Ausgabe zurück.

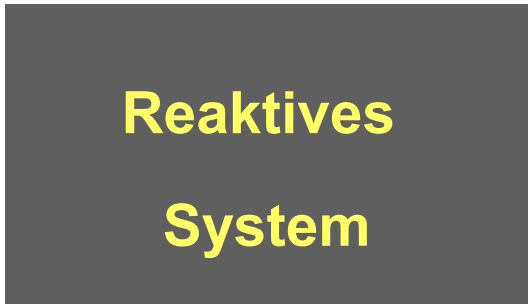
Terminierende Systeme: Qualitätssicherung

- Das Testobjekt ist eine **(partielle) Funktion** zwischen Eingabe- und Ausgabewertedomäne, die (komplexe) Eingabedaten in (komplexe) Ausgabedaten **transformiert**.
- Die Ausgabe ist zumeist **unabhängig** vom internen Zustand des Software-Systems.
- Aufgabe der funktionalen Qualitätssicherung ist es, die Korrektheit anhand von möglichst vielen **Eingabe-Ausgabe-Werte-Paaren** zu prüfen.

Reaktive Systeme

???

???



Annahme nun: Das Software-(Teil-)System zeigt **fortlaufend** („*für immer*“) für **asynchron** („*wild durcheinander*“) auftretende Eingaben (**Aktionen**) passende Ausgaben (**Reaktionen**), die vom internen **Betriebszustand** („*der bisherigen Vorgeschichte*“) abhängen.

Beispiele: Reaktive Systeme

- Betriebssysteme und Treiber
- Kommunikationsprotokolle
- Eingebettete Steuerungs- und Regelungssysteme
- Navigations-, Tracking- und Monitorkomponenten
- …

Reaktive Systeme: Weitere Merkmale

- **Reaktivität:** Wesentlich sind kontinuierliche Aktions-Reaktions-Abfolgen, die Eingabe- und Ausgabedaten und deren Transformation selbst sind hingegen zumeist einfach strukturiert („atomare Stimuli ohne besonderen Inhalt“).
- **Asynchrone Interaktionen:** Umgebung mit beliebig vielen und potentiell dynamisch wechselnden „Kommunikationspartnern“ (Menschen, Sensoren/Aktoren, andere Teile der Software).
- **Nichtterminierung:** ist der Normalfall, Terminierung stellt zumeist ein unerwünschtes Fehlverhalten da.
- **Nichtdeterminismus:** ist beabsichtigt und dient z.B. zur Unterspezifikation unklaren/unbekannten (Umgebungs-)Verhaltens.
- **Nebenläufigkeit, Verteilung, Mobilität:** von (Teil-)Systemen sind wesentlich.

Reaktive Systeme: Qualitätssicherung

- Das Testobjekt besteht aus mehreren nebenläufigen Teilen, die jeweils nicht-endliche und sich potentiell gegenseitig beeinflussende Aktions-Reaktions-Abfolgen ausführen.
- Nahezu jede Eingabe führt in Abhängigkeit vom derzeitigen Systemzustand zu einem neuen Systemzustand.
- Nahezu jede Ausgabe ist abhängig von der Eingabe und dem derzeitigen Systemzustand.
- Wie wird ein solches Verhalten spezifiziert und während der Qualitätssicherung mit einer Implementierung verglichen?
- Bisherige Ansätze zur Qualitätssicherung durch Vergleich von Eingabe-Ausgabe-Werte-Paaren sind dafür unzureichend.

Prozesstheorie

- Programme mit den zuvor beschriebenen Eigenschaften werden auch als **Prozesse** bezeichnet.
- Die **Prozesstheorie** befasst sich mit der formalen Spezifikationen und Analyse des Verhaltens reaktiver Software-Systeme.
- Eine **Prozess-Algebra** ist ein mathematischer Kernkalkül zur Spezifikation und zum Vergleich des Verhaltens nebenläufiger Prozesse.

Prozess-Algebra: Grundlagen

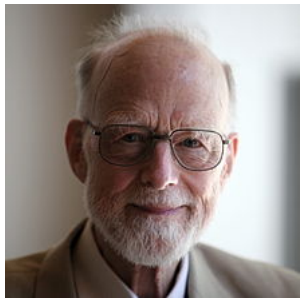
Eine Prozessalgebra definiert eine formale Sprache $\mathcal{L} = (\mathcal{P}, \rightarrow, \equiv, \simeq)$, bestehend aus:

- $\mathcal{P}$: **Syntax** von Prozessen (Prozesstermen), zumeist definiert durch eine (kontextfreie) Grammatik.
- $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$: **operationelle Semantik** von Prozessen, definiert durch Regeln für mögliche Zustandswechsel als Prozessterm-Transformationen.
- $\equiv \subseteq \mathcal{P} \times \mathcal{P}$: **strukturelle Kongruenz**, definiert durch syntaktische Regeln für die Äquivalenz von Prozesstermen.
- $\simeq \subseteq \mathcal{P} \times \mathcal{P}$: **Verhaltensäquivalenz**, definiert durch semantische Regeln für die Äquivalenz von Prozesstermen.

Prozess-Algebra: Historie

λ-Calculus (Alonzo Church, 1930)
Sequentielle I/O-Programme

CCS (Robin Milner, ~1980)
Reaktive nebenläufige Prozesse



CSP (C. A. R. Hoare, ~1980)
Reaktive nebenläufige Prozesse

Prozess-Algebren: Anmerkungen

- CCS = Calculus of Communicating Systems
- CSP = Communicating Sequential Processes
- Beide Kalküle weisen sehr viele Gemeinsamkeiten auf, die Unterschiede verstecken sich vor allem in einigen „technischen“ Details.
- Wie betrachten nachfolgend CCS.

UNIVERSITÄT SIEGEN

Einführung in CCS

Prozesse

Wir betrachten reaktive Software-Systeme, die aus einer **Menge von nebenläufigen, interagierenden Prozessen** bestehen.

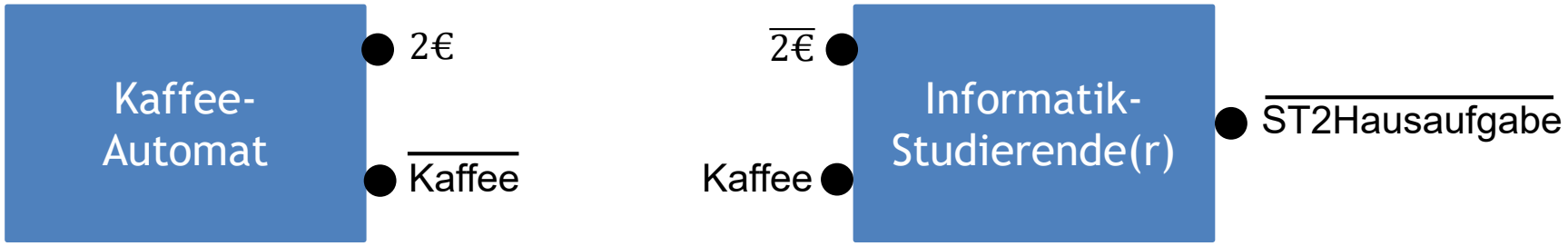
Aus einer Black-Box-Perspektive betrachtet, besteht ein Prozess aus:

- Einem Prozessnamen P.
- Einer Prozessschnittstelle.

Prozessschnittstelle

- Die Prozessschnittstelle definiert eine Menge von **Kommunikations-Ports** (**Kanäle**), über die der Prozess mit seiner Umgebung interagiert.
- Für jeden Port wird der Name einer **Aktion** (**Nachricht**) sowie die **Richtung** der Kommunikation (**eingehende** oder **ausgehende** Nachricht) angegeben.
- Die Menge der Aktionsnamen ist global einheitlich für das gesamte System definiert.

Beispiel: Prozesse



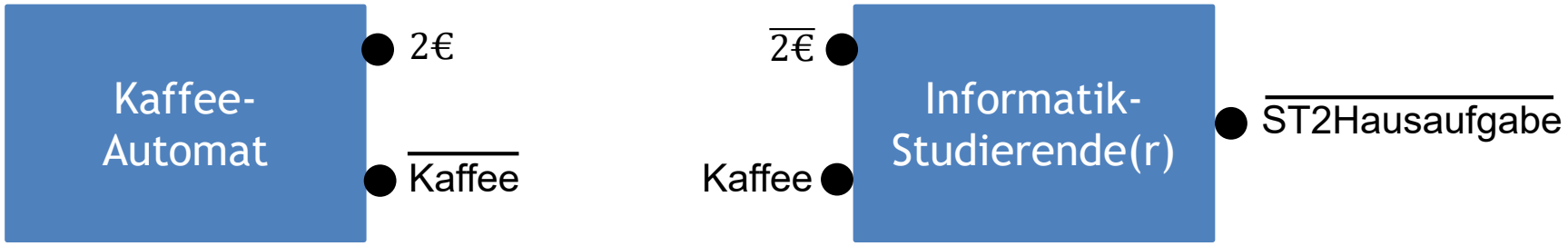
Dieses reaktive System besteht aus:

- Zwei Prozessen: „Kaffee-Automat“ mit zwei Ports und „Informatik-Studierende(r)“ mit drei Ports.
- Drei Aktionen: „2€“, „Kaffee“ und „ST2Hausaufgabe“.

Beispiel: Erläuterungen

- Prozesse werden hier als Rechtecke dargestellt, die mit dem Prozessnamen P beschriftet sind.
- Ports werden als schwarze Kreise am Rand des Prozessrechteckes dargestellt, die mit dem Aktionsnamen a beschriftet sind.
  - Beschriftung mit a bedeutet, dass a eine eingehende Aktion (Empfangsaktion) des Prozesses ist.
  - Beschriftung mit $\bar{a}$ bedeutet, dass a eine ausgehende Aktion (Sendeaktion) des Prozesses ist.

Beispiel: Prozessschnittstellen

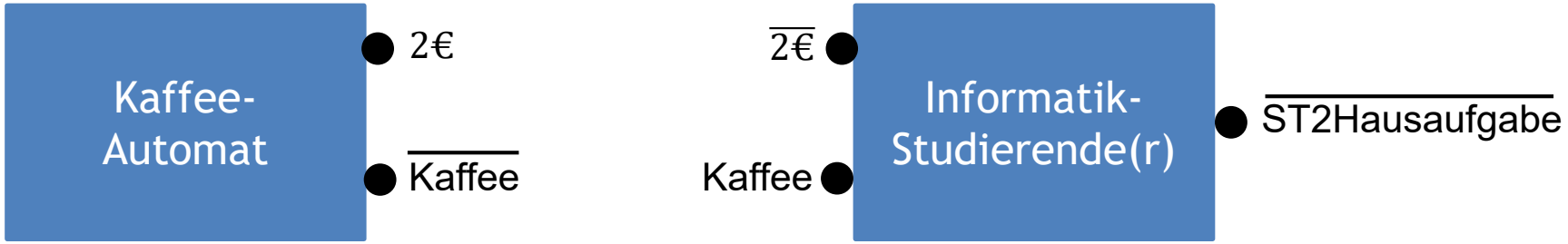


- Ein Kaffee-Automat kann $2€$-Stücke entgegennehmen und Becher mit Kaffee ausgeben.
- Eine Informatik-Studierende kann $2€$-Stücke ausgeben, Becher mit Kaffee konsumieren und ST-2-Hausaufgaben produzieren.

Prozessverhalten

- Die Prozessschnittstelle beschreibt das mögliche Verhalten eines Prozesses P als die Menge von Aktionen, die der Prozess **potenziell** ausführen kann.
- Ohne weitere Angaben zum **Prozessverhalten** sind verschiedene Interpretationen möglich:
  - Der Prozess kann von sich aus erstmal gar nichts tun.
  - Der Prozess kann von sich aus jederzeit jede seiner Aktionen in beliebiger Reihenfolge ausführen.
  - …

Beispiel: Prozessverhalten



- Der Kaffee-Automat soll nach der Eingabe eines 2€-Stücks direkt danach genau einen Becher mit Kaffee ausgeben.
  Was passiert, wenn während des Füllens eines Bechers ein weiteres 2€-Stück eingeworfen wird? Was passiert, wenn der Vorrat an Kaffee aufgebraucht ist? …
- Eine Informatik-Studierende braucht mindestens einen Becher Kaffee, um die ST-2-Hausaufgabe fertigzustellen.
  Was passiert, wenn sie die Hausaufgaben mit einem Kommilitonen zusammen bearbeitet und sich beide einen Kaffee teilen? …

Prozessverhalten

- Wir gehen davon aus, dass ein Prozess nur dasjenige Verhalten aufweisen kann, das explizit spezifiziert worden ist.
- CCS definiert Syntax und Semantik von **Prozess-Termen** zur formalen Spezifikation von Prozessverhalten über einer Aktionsmenge.
- Wir führen zunächst informell die Syntax und intuitive Semantik der CCS-Konstrukte ein.

Der leere Prozess

Syntax: 0

Semantik:

- Prozess, der keine Aktion ausführt.
- Prozess, der direkt terminiert.

(sprich „nil“)

Beispiel: Der leere Prozess

Kaputter-Kaffee-Automat := $\mathbf{0}$

Fauler-Informatik-Studierender := $\mathbf{0}$

- Der kaputte Kaffee-Automat nimmt niemals ein Geldstück entgegen und gibt niemals einen Becher Kaffee aus.
- Der faule Informatik-Studierende gibt niemals $2€$ aus, gibt niemals eine ST-2-Hausaufgabe ab und trinkt niemals einen Becher Kaffee (?)

Der sequenzielle Prozess

Syntax: $a\,.\,P$

Semantik:

- Prozess, der erst Aktion a ausführt und sich dann wie Prozess P verhält.
- Aktion a ist Teil der Schnittstelle des beschriebenen Prozesses.
- Abfolgen (Sequenzen) mehrerer Aktionen möglich.

Beispiel: Der sequenzielle Prozess

$$\text{Sehr-Billiger-Kaffee-Automat} := 2€ \,.\, \overline{\text{Kaffee}} \,.\, \mathbf{0}$$

$$\text{Billiger-Kaffee-Automat} := 2€ \,.\, \overline{\text{Kaffee}} \,.\, 2€ \,.\, \overline{\text{Kaffee}} \,.\, \mathbf{0}$$

$$\text{Fleißige-Studierende} := \overline{\text{ST2Hausaufgabe}} \,.\, \mathbf{0}$$

- Der sehr billige Kaffee-Automat kann ein einziges Mal Geld annehmen und einen Becher Kaffee ausgeben, bevor er kaputt geht.
- Der billige Kaffee-Automat schafft immerhin zwei Vorgänge, bevor er kaputt geht.
- Die fleißige Studierende erledigt die ST-2-Hausaufgabe allein und ohne einen einzigen Kaffee.

Anmerkungen: Der sequenzielle Prozess

- Das Symbol "." wird **Präfix-Operator** genannt.
- Aktionsausführungen sind stets **atomar**, d.h. die Aktion wird vollständig (in „Nullzeit“) oder gar nicht ausgeführt.
- Aktionssequenzen dürfen (syntaktisch) beliebig lang, aber nur endlich lang sein.

Der benannte Prozess

Syntax: Pname := P

Semantik:

- Der Prozess mit dem Namen „Pname“ wird spezifiziert durch den Prozessterm P.
- Diese Syntax wurde in den vorherigen Beispielen schon stillschweigend genutzt.
- Bisher noch nicht betrachtet: Prozessnamen können auch in Prozesstermen auftreten.

Beispiel: Der benannte Prozess

$$
\begin{aligned}
&\text{Sehr-Billiger-Kaffee-Automat} &&:= \text{Kaufvorgang} \,.\, \mathbf{0} \\
&\text{Kaufvorgang} &&:= 2€ \,.\, \overline{\text{Kaffee}} \,.\, \mathbf{0} \\
&\text{Informatik-Studierende} &&:= \text{Pause} \,.\, \overline{\text{ST2Hausaufgabe}} \,.\, \mathbf{0} \\
&\text{Pause} &&:= \overline{2€} \,.\, \text{Kaffee} \,.\, \mathbf{0}
\end{aligned}
$$

- Die Prozessspezifikation des sehr billigen Kaffee-Automaten hat sich nicht geändert, aber der einmalige Teilprozess „Kaufvorgang“ wurde nun benannt und in einen eigenen Prozessterm ausgelagert.
- Die Informatik-Studierende braucht nun erstmal eine Kaffeepause, bevor die ST-2-Hausaufgabe erledigt werden kann.

Anmerkungen: Der benannte Prozess

- Es können für beliebig viele, aber nur endlich viele Prozessnamen entsprechende Prozessterme hinterlegt werden.
- Die Prozessnamen müssen global eindeutig sein und müssen von den Aktionsnamen verschieden sein.
- Häufige Konvention: Prozessnamen beginnen mit einem Großbuchstaben, Aktionsnamen mit einem Kleinbuchstaben (im Beispiel nicht ganz durchgehalten).

Der rekursive Prozess

Syntax: Pname := P

Semantik:

- Der Prozess mit dem Namen „Pname“ wird spezifiziert durch den Prozessterm P.
- Der Prozessterm P kann Prozessnamen enthalten („Prozessaufrufe“).
- Insbesondere kann P wieder „ Pname“ enthalten und somit rekursiv definiert sein.

Beispiel: Der rekursive Prozess

Kaffee-Automat	:= 2€ . $\overline{\text{Kaffee}}$. Kaffee-Automat
1xGratis-Kaffee-Automat	:= $\overline{\text{Kaffee}}$. Kaffee-Automat
Betrugs-Kaffee-Automat	:= 2€ . Betrugs-Kaffee-Automat
Informatik-Studierender	:= $\overline{\text{2€}}$. Kaffee . Informatik-Studierender

- Der „Kaffee-Automat“ kann beliebig viele Kaufvorgänge abwickeln.
- Der „1xGratis-Kaffee-Automat“ gibt einmalig einen Becher Kaffee gratis aus, danach verhält er sich wie der normale „Kaffee-Automat“.
- Der „Betrugs-Kaffee-Automat“ nimmt immer weitere Geldstücke entgegen, ohne jemals Kaffee auszugeben.
- Der Informatik-Studierende konsumiert einen Kaffee nach dem anderen…

Anmerkungen: Der Rekursive Prozess

- Durch rekursive Prozessdefinitionen können nichtterminierende Prozesse beschrieben werden.
- Im Beispiel fehlt dem Informatik-Studierenden die Wahlmöglichkeit, irgendwann den Kaffeekonsum zu unterbrechen, um die ST-2-Hausaufgaben anzugehen.

Der auswählende Prozess

Syntax: $P + Q$

Semantik:

- Prozess, der sich entweder wie der Prozess P oder wie der Prozess Q verhält.

Beispiel: Der Auswählende Prozess

Fuzzy-Kaffee-Automat := (Gratis + Betrug) . Fuzzy-Kaffee-Automat

1xGratis-Kaffee-Automat := $\overline{\text{Kaffee}}$. **0**

Betrug := 2€ . **0**

Informatik-Studierender := (Pause + $\overline{\text{ST2Hausaufgabe}}$. **0**) . Informatik-Studierender

Pause := $\overline{\text{2€}}$. Kaffee . **0**

Beispiel: Der Auswählende Prozess

- Der „Fuzzy-Kaffee-Automat“ gibt entweder einen Gratis-Kaffee aus oder nimmt ein Geldstück entgegen, ohne einen Becher Kaffee auszugeben.
- Die Informatik-Studierende macht entweder eine Kaffeepause oder geht die ST-2-Hausaufgaben an.

Anmerkungen: Der Auswählende Prozess

- Das Symbol „+“ wird **Auswahloperator** (bzw. „choice“ oder „branch“) genannt.
- Die Auswahl des Folgeverhaltens erfolgt nichtdeterministisch und es gibt keine Fairness-Annahmen.
- Es können beliebig viele, aber nur endlich viele Prozesse zur Auswahl aufgelistet werden.
- Die in den Beispielen verwendeten Klammerungen von Teilprozessen hat die übliche Bedeutung. Dabei ist zu beachten, dass nach der Terminierung eines Teilprozesses durch Erreichen des leeren Prozesses nur dieser Teilprozess terminiert und der Ablauf dann im Gesamtprozess eine Ebene „weiter oben“ fortgesetzt wird.

Anmerkungen: Der Auswählende Prozess

- Bisher noch nicht spezifiziert: Die Interaktion zwischen dem Kaffee-Automaten und dem Informatik-Studierenden ist weder sequenziell noch eine Auswahl.
- Beide Prozesse verlaufen vielmehr eigenständig (**nebenläufig**) voneinander, können (bzw. müssen) sich aber bei Bedarf **synchronisieren**, um zu **interagieren**.

Der nebenläufige Prozess

Syntax: $P \mid Q$

Semantik:

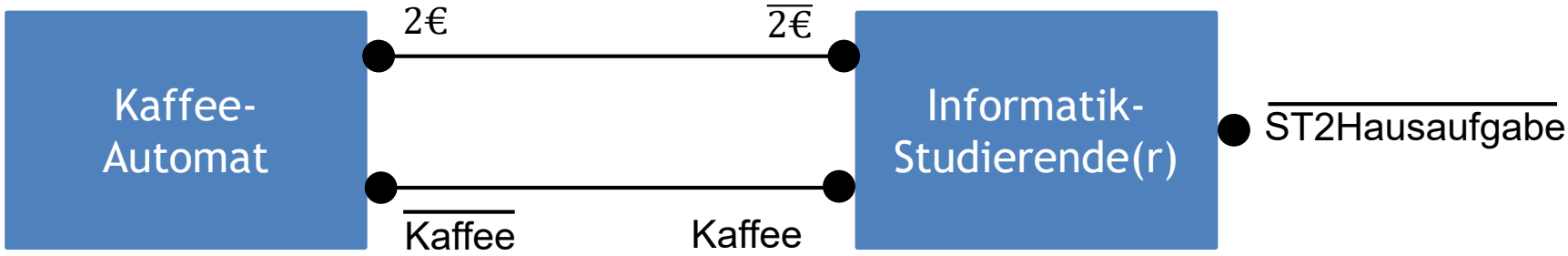
- Prozess, in dem P und Q unabhängig voneinander fortfahren können.
- Dabei können P und Q über *kompatible* Ports miteinander kommunizieren.

Kompatible Ports

Die Ports zweier Prozesse sind **kompatibel**, wenn:

- sie mit dem gleichen Aktionsnamen a beschriftet sind und
- die Kommunikationsrichtungen entgegengesetzt sind, d.h. ein Port mit a und der andere Port mit $\bar{a}$ beschriftet ist.

Beispiel: Kompatible Ports



- Die kompatiblen Ports der beiden Prozesse „Kaffee-Automat“ und „Informatik-Studierende(r)“ sind jeweils mit Linien („Kanälen“) verbunden.
- Der Port „ST2Hausaufgabe“ von „Informatik-Studierende(r)“ hat hingegen keinen kompatiblen Port im „Kaffee-Automat“.

Beispiel: Der nebenläufige Prozess

Kaffee-Automat := 2€ . $\overline{\text{Kaffee}}$. Kaffee-Automat

Informatik-Studi := ($\overline{\text{2€}}$. Kaffee . **0** + $\overline{\text{ST2Hausaufgabe}}$. **0**) . Informatik-Studi

Informatik-Studium := Informatik-Studi | Kaffee-Automat

- Gesamtsystem „Informatik-Studium“ ergibt sich durch „Parallelschalten“ der Prozesse „Informatik-Studi“ und „Kaffee-Automat“.
- Der „Informatik-Studi“ macht entweder mit Hilfe des Kaffee-Automaten eine Pause oder produziert selbständig eine ST-2-Hausaufgabe.

Anmerkungen: Der Nebenläufige Prozess

- Das Symbol „|“ wird (paralleler) **Kompositionsoperator** genannt.
- Es können beliebig viele, aber nur endlich viele nebenläufige Prozesse aufgelistet werden.
- Achtung: Tatsächlich wird in CCS die synchrone Ausführung von kompatiblen Aktionen nebenläufiger Prozesse nicht unbedingt erzwungen (dazu gleich mehr).

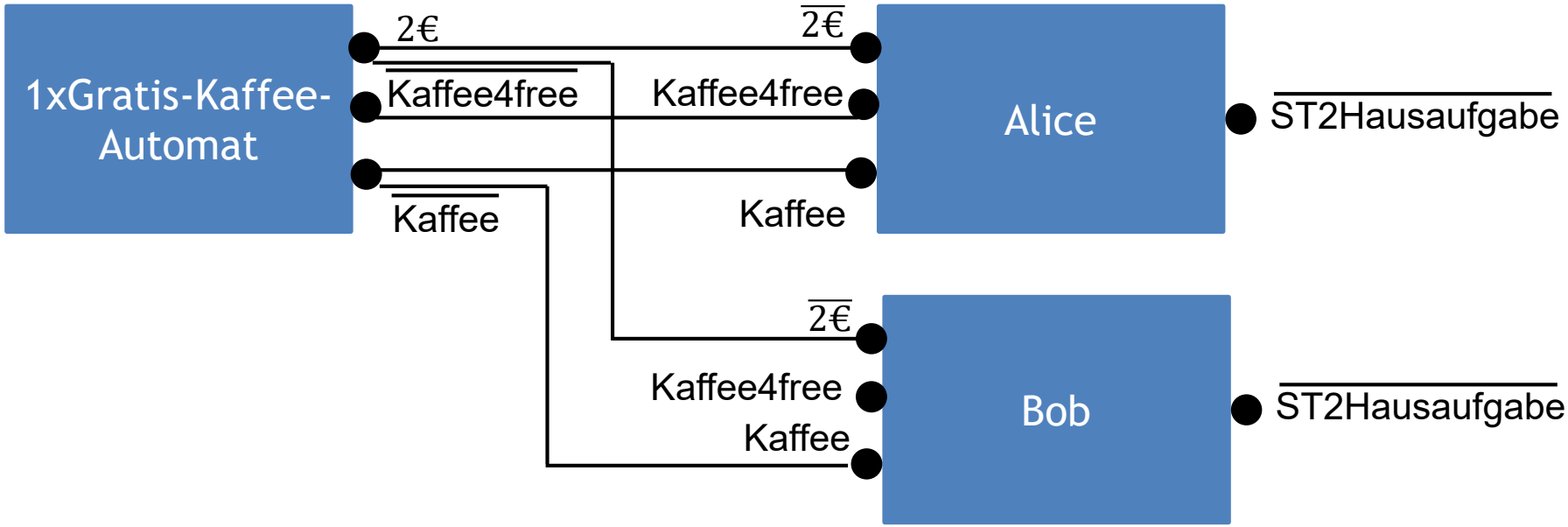
Der Eingeschränkte Prozess

Syntax: $P \setminus L$

Semantik:

- Prozess, der sich wie P verhält, wobei eine (Teil-)Menge L der Aktionenmenge der Ports von P außerhalb von P verborgen bleibt.
- Prozess P ist der *Scope* der Aktionenmenge L.

Beispiel: Der Eingeschränkte Prozess



- Alice und Bob konkurrieren um den Gratis-Kaffee.

Beispiel: Der Eingeschränkte Prozess

Kaffee-Automat	:= $2€ . \overline{\text{Kaffee}}$. Kaffee-Automat
1xGratis-Kaffee-Automat	:= $\overline{\text{Kaffee4free}}$. Kaffee-Automat
Alice	:= Kaffee4free . $\overline{\text{ST2Hausaufgabe}}$. **0**
Bob	:= (Kaffee4free . **0** + $\overline{2€}$. Kaffee . **0**) . $\overline{\text{ST2Hausaufgabe}}$. **0**
ST2Deadline	:= ((Alice | 1xGratis-Kaffee-Automat) \ {Kaffee4free}) | Bob

- Alice versteckt die „Kaffee4free“-Aktion vor Bob.

Anmerkungen: Der Eingeschränkte Prozess

- Das Symbol „\“ wird **Restriktionsoperator** genannt.

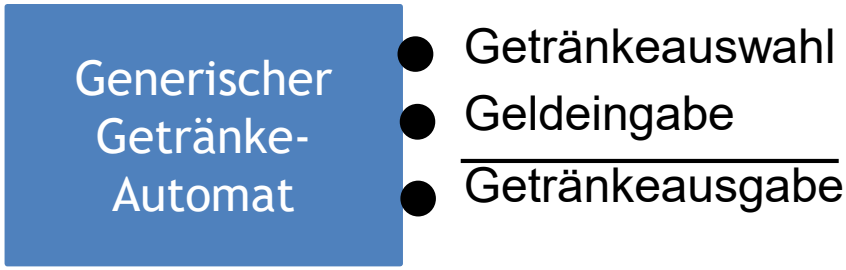
Der Umbenannte Prozess

Syntax: $\boldsymbol{P}[f]$

Semantik:

- Prozess, der sich wie $\boldsymbol{P}$ verhält, wobei Aktionen a in $f(a)$ umbenannt werden.
- Die Umbenennungsfunktion f muss dabei bestimmte Anforderungen erfüllen (siehe später).

Beispiel: Der Umbenannte Prozess



Kaufvorgänge für verschiedene Getränkesorten folgen stets dem gleichen Prozessschema, in denen sich nur konkrete Aktionsnamen unterscheiden.

Beispiel: Der Umbenannte Prozess

Kaufvorgang	:= Getränkeauswahl . Geldeingabe . $\overline{\text{Getränkeausgabe}}$. **0**
Kaffee-Kaufvorgang	:= Kaufvorgang[Kaffee-Auswahl/Getränkeauswahl, 2€/Geldeingabe, Getränkeausgabe/Kaffee]
Club-Mate-Kaufvorgang	:= Kaufvorgang[Club-Mate-Auswahl/Getränkeauswahl, 3€/Geldeingabe, Getränkeausgabe/Club-Mate]
	…
Getränke-Automat	:= (Kaffee-Kaufvorgang + Club-Mate-Kaufvorgang + …) . Getränke-Automat

Anmerkungen: Der Umbenannte Prozess

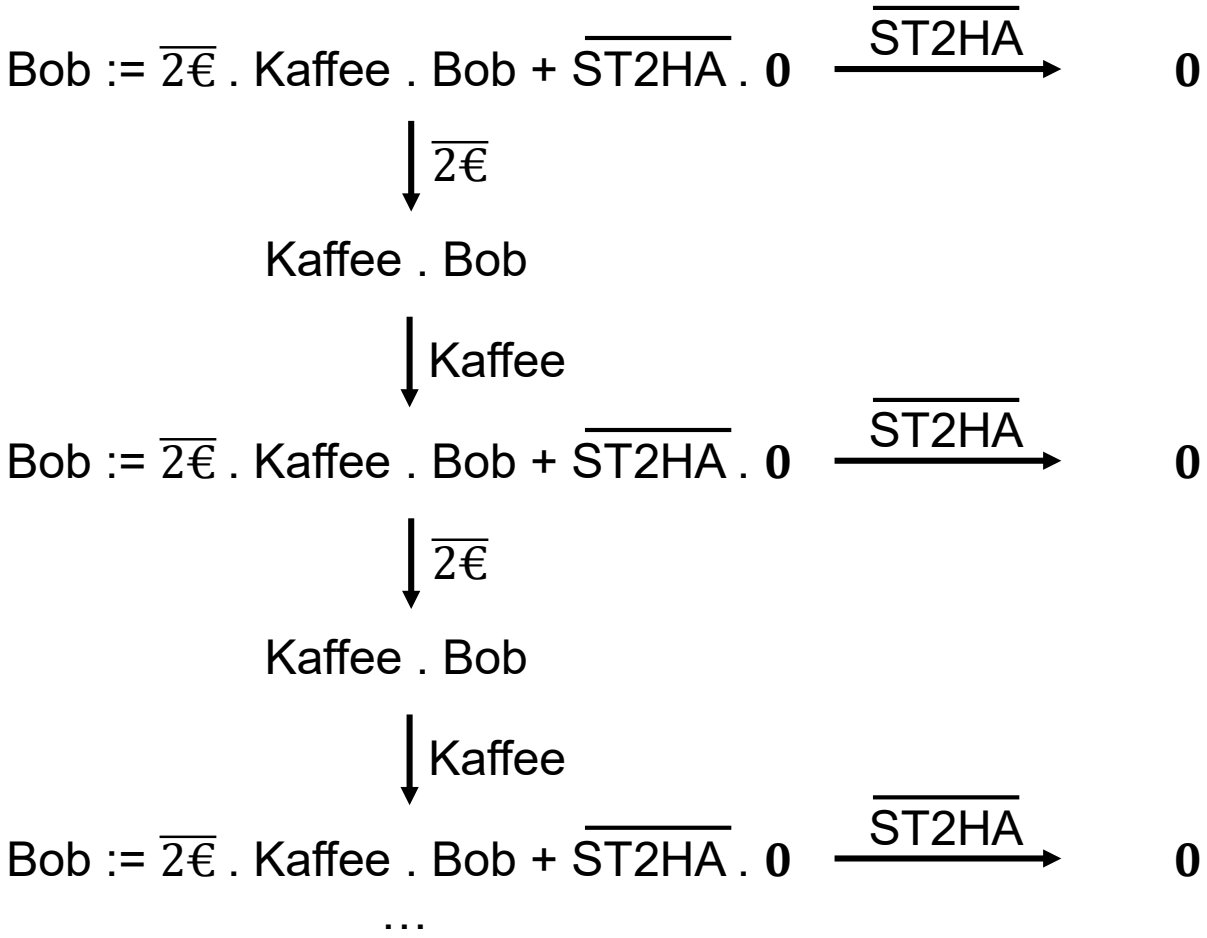
- Die Symbolkombination „$[a/b]$“ wird **Umbenennungsoperator** (Renaming, Relabeling) genannt.
- Dabei ist a der neue Aktionsname, der den alten Aktionsnamen b ersetzt.
- Damit haben wir alle Operatoren von CCS kennengelernt.
- CCS ist **Turing-vollständig**, d.h. es kann eine Turing-Maschine in CCS simuliert werden.

Das Verhalten von CCS-Prozessen

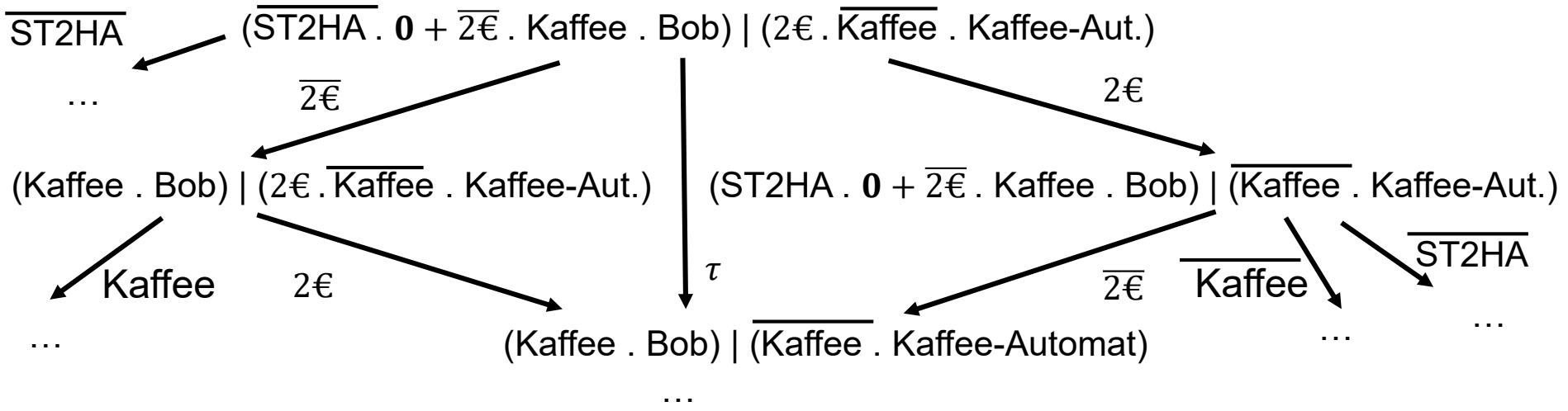
Das Verhalten eines Prozesses „*Pname*“ ist definiert durch die **strukturelle operationelle Semantik** von CCS:

- Die syntaktische Struktur des Prozesstermes $\boldsymbol{P}$ des Prozesses „Pname“ definiert den gegenwärtigen **Zustand** des Prozesses.
- Genauer: Die im Term $\boldsymbol{P}$ vorkommenden CCS-Operatoren, Prozessnamen und Aktionsnamen bestimmen eindeutig die Menge möglicher Aktionsschritte.
- Je nach tatsächlich ausgeführter Aktion a ergibt sich der Folgezustand des Prozesses durch Transformation des Prozessterms $\boldsymbol{P}$ in den Prozessterm $\boldsymbol{P'}$.

Beispiel: Das Verhalten von CCS-Prozessen



Beispiel: Das Verhalten von CCS-Prozessen



Verhalten der beiden Prozesse „Bob“ und „Kaffee-Automat“ parallel geschaltet (Ausschnitt).

…

Anmerkungen zum Beispiel

- Die beiden Prozesse „Kaffee-Automat“ und „Bob“ führen entweder unabhängig voneinander in zwei getrennten Schritten die Ein-/Ausgabe-Aktion „$2€$“ bzw. „$\overline{2€}$“ aus oder gemeinsam in einem synchronen Schritt.
- Gemeinsame Schritte sind statt mit der konkreten Aktion mit dem speziellen Aktionsnamen „τ“ beschriftet, der von außen nicht beobachtbar ist.

(sprich „tau“)

Anmerkungen zum Beispiel

- Durch das Verstecken gemeinsam ausgeführter Aktionen wird Kommunikation in CCS somit stets auf genau einen Sender- und einen Empfängerprozess beschränkt.
- Im Beispiel könnten andernfalls **mehrere** Empfänger das **gleiche** Geldstück zugleich in Empfang nehmen.

Anmerkungen zum Beispiel

- Das Beispiel zeigt weiterhin, dass Kommunikation in CCS nicht **blockierend** ist, da beide Prozesse auch stets unabhängig voneinander ihre Abläufe fortsetzen können. Den Grund dafür werden wir später sehen.
- Diese „halben“ Kommunikationsschritte könnten durch Verwendung des Restriktionsoperators ebenfalls nach außen versteckt werden (siehe später).

UNIVERSITÄT SIEGEN

Formale Syntax und Semantik von CCS und LTS

Formale Definition von CCS

- Bisher: Informelle Einführung der Syntax und Semantik von CCS-Prozesstermen und den darin verwendeten Operatoren.
- Nachfolgend lernen wir eine präzise Definitionen der Syntax und Semantik von CCS-Prozesstermen kennen.
- Auf dieser Grundlagen werden einige semantische Details verschiedener Operatoren noch besser verständlich.

Aktionsnamen

- Mit $\mathcal{A}$ bezeichnen wir die global für alle Prozesse des betrachteten Systems definierte, abzählbar unendlich große Menge von **Aktionsnamen** für Eingabeaktionen.
- Mit $\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$ bezeichnen wir die Menge der zu $\mathcal{A}$ **komplementären Aktionsnamen** (Co-Namen) für Ausgabeaktionen.
- Offensichtlich gilt: $\bar{\bar{a}} = a$.

Aktions-Labels und Aktionen

- Mit $\mathcal{L} = \mathcal{A} \cup \bar{\mathcal{A}}$ bezeichnen wir die Menge der **Aktions-Labels** und Elemente aus $\mathcal{L}$ bezeichnen wir zumeist mit den Symbolen $a, b, \ldots$
- Mit $Act = \mathcal{L} \cup \{\tau\}$ bezeichnen wir die Menge aller **Aktionen** und ein Element aus Act bezeichnen wir mit α.

Prozessnamen

- Mit $\mathcal{K}$ bezeichnen wir die global für das betrachtete Systems definierte, abzählbar unendlich Große Menge von **Prozessnamen**.
- Elemente aus $\mathcal{K}$ werden auch als (Prozess-)Konstanten bezeichnet.
- Die spezielle Prozesskonstante $\mathbf{0}$ ist hingegen nicht in der Menge $\mathcal{K}$, da für den leeren Prozess keine Prozessdefinition angegeben werden kann.

UNIVERSITÄT SIEGEN

CCS-Prozessterme: Formale Syntax

Mit $\mathcal{P}$ bezeichnen wir die Menge aller CCS-Terme über den Mengen Act und $\mathcal{K}$, die durch die folgende Grammatik definiert ist:

$$P, Q ::= \mathrm{K} \mid \alpha.P \mid \sum_{i \in I} P_i \mid P|Q \mid P[f] \mid P \setminus L$$

Anmerkungen zur Formalen Syntax

- $K \in \mathcal{K}$ ist ein Prozessname, für den eine Prozessdefinition der Form $K \coloneqq P$ existiert.
- $\alpha \in Act$ ist eine Aktion.
- I ist eine Menge von Prozess-Indizes.
- $f: Act \to Act$ ist eine Umbenennungsfunktion, für die gilt: $f(\tau) = \tau$ und $\forall a \in \mathcal{L}$: $f(\bar{a}) = \overline{f(a)}$.
- $L \subseteq \mathcal{L}$ ist eine Menge von Labels.

Anmerkungen zur Formalen Syntax

- Der leere Prozess $\mathbf{0}$ ist eine Kurzform für die leere Summe: $\sum_{i \in \emptyset} P_i$
- Die bisher verwendete Schreibweise $P_1 + P_2$ für die Auswahl zwischen zwei Prozessen ist eine Kurzform für die Summe: $\sum_{i \in \{1,2\}} P_i$

Operatorpräzedenzen

- „.“ bindet stärker als „+“
- „+“ bindet stärker als „|“
- Für den Wirkungsbereich von „\“ und „[…]“ sollten hingegen immer Klammern gesetzt werden.

Operationelle Semantik

- Die operationelle Semantik von CCS basiert auf **Labeled Transition Systems (LTS).**
- Ein LTS besteht wie ein Zustandsautomat aus einer Menge S von **Zuständen** und **Transitionen** der Form:

$$s \xrightarrow{a} s'$$

- Transitionen sind mit Aktionen $a \in A$ aus einer Aktionenmenge A (**Operationen**) beschriftet und definieren die möglichen Schritte (**Zustandsübergänge**) des Systems (bzw. des Prozesses).

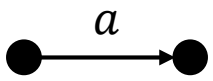
Definition: Labeled Transition System (LTS)

Ein Labeled Transition System (LTS) ist ein Tripel $(S, A, \rightarrow)$ bestehend aus:

- einer *abzählbar unendlichen* Menge S von **Zuständen**,
- einer *abzählbar unendlichen* Menge A von **Aktionen** und
- einer Menge $\rightarrow \subseteq S \times A \times S$ von **Transitionen.**

Notationskonventionen für LTS

- Falls $(s, a, s') \in \rightarrow$, dann schreiben wir $s \xrightarrow{a} s'$.
- Falls kein s' existiert, sodass $(s, a, s') \in \rightarrow$ gilt, dann schreiben wir $s \overset{a}{\not\mapsto}$
- Für die graphische Darstellung von (Ausschnitten eines) LTS werden Zustände meistens als ausgefüllte Punkte dargestellt:



Vergleich: LTS vs. Zustandsautomaten

- Die Zustandsmenge, Aktionsmenge und damit auch Transitionenmenge von Zustandsautomaten sind **endlich** (die von LTS hingegen abzählbar unendlich).
- Zustandsautomaten haben einen ausgezeichneten **Startzustand** (LTS in der Regel nicht).
- Zustandsautomaten sind **zusammenhängend**, d.h., jeder Zustand ist über mindestens einen Pfad vom Startzustand aus erreichbar (LTS u. U. nicht).

Anmerkungen: LTS als Semantisches Modell

- LTS dienen im Gegensatz zu Zustandsautomaten nicht als Modellierungssprache, sondern als semantisches Modell, d.h. als formale Grundlage zur Repräsentation von Sprachsemantiken.
- Zur Abbildung der Semantik von *Turing-vollständigen* Sprachen ist es essentiell, abzählbar unendlich viele Zustände, Aktionen und Transitionen zuzulassen.

Beispiel: LTS-Semantik

Die Semantik der Operation „$Successor$“ auf der Menge der natürlichen Zahlen lässt sich durch ein LTS $(\mathbb{N}_0, \{succ\}, \rightarrow)$ definieren:

- Zustandsmenge S ist die Menge der natürlichen Zahlen $\mathbb{N}_0$.
- Aktionsmenge ist die $succ$-Operationen.
- Die Menge der Transitionen lässt sich beschreiben durch die Regel $\{n \overset{succ}{\rightarrow} n+1 \mid n \in \mathbb{N}_0\}$

Beispiel: LTS-Semantik

Die Semantik der Operation „$Goto$“ auf der Menge der natürlichen Zahlen lässt sich durch ein LTS $(\mathbb{N}_0, \mathbb{N}_0, \rightarrow)$ definieren:

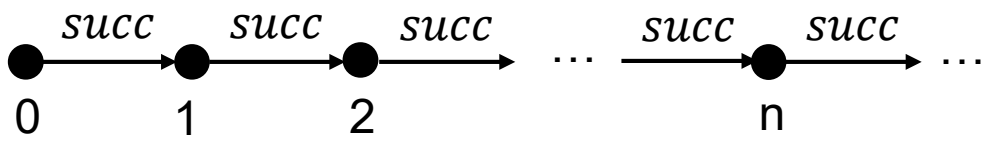
- Zustandsmenge S ist die Menge der natürlichen Zahlen $\mathbb{N}_0$.
- Aktionsmenge A ist die Menge der natürlichen Zahlen $\mathbb{N}_0$.
- Die Menge der Transitionen lässt sich beschreiben durch die Regel $\{n \xrightarrow{n'} n' | n, n' \in \mathbb{N}_0\}$

Eigenschaften von LTS

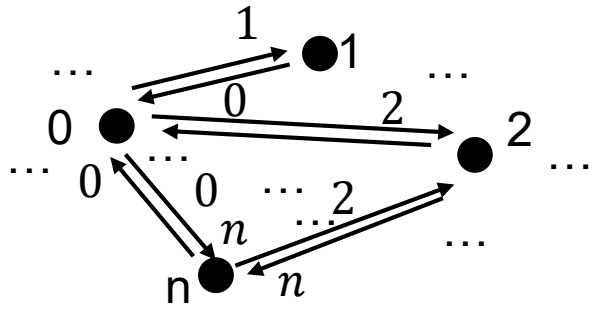
- Ein LTS ist **endlich verzweigend**, falls jeder Zustand $s \in S$ nur endlich viele ausgehende Transitionen $s \xrightarrow{a} s'$ hat.
- Ein LTS ist **regulär**, falls von jedem Zustand $s \in S$ aus nur endlich viele Zustände $s' \in S$ über beliebige Pfade erreicht werden können.
- Ein LTS ist **endlich**, falls es endlich verzweigend ist und keine nicht-endlichen Pfade enthält.

Beispiel: Eigenschaften von LTS

- Das LTS für $Successor$ ist endlich verzweigend, aber nicht regulär und nicht endlich.



- Das LTS für $Goto$ ist nicht endlich verzweigend, nicht regulär und nicht endlich.



Strukturelle Operationelle Semantiken

- Eine strukturelle operationelle Semantik (SOS) definiert die Semantik einer formalen Sprache als LTS.
- Das Besondere an einer SOS ist, dass die Definition der Zustände und Transitionen des LTS durch rein syntaktische (strukturelle) Regeln erfolgt:
  - Die Menge der Zustände ist die Menge aller Prozessterme $\mathcal{P}$, die gemäß der formalen Syntax der Sprache gebildet werden können.
  - Die Menge der Transitionen $P \xrightarrow{a} P'$ wird durch kontextfreie Inferenzregeln definiert, die sich nur auf strukturelle Eigenschaften des aktuellen Zustands (Prozessterm P) beziehen, um den Folgezustand (Prozessterm P') für eine Aktion a abzuleiten.

Format von Inferenzregeln

$$\textit{[Regelname]}\ \frac{\text{[Prämisse über } P, P'\text{]}}{P \xrightarrow{a} P'}\ \textit{[weitere Bedingungen]}$$

- Die *Prämisse* kann Eigenschaften über der syntaktischen Struktur von P und P' sowie (zuvor abzuleitende) Transitionen fordern.
- Die Schlussfolgerung unter dem Strich wird *Conclusio* genannt.

SOS für CCS-Prozessterme

Die operationelle Semantik der Menge $\mathcal{P}$ aller CCS-Prozessterme über den Mengen Act und $\mathcal{K}$ ist definiert durch das LTS

$$(\mathcal{P}, Act, \rightarrow)$$

wobei für die Relation $\rightarrow$ die nachfolgenden Inferenz-Regeln gelten.

Präfix-Regel

$$[\text{pre}]\ \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

- Eine Aktion α, die in einem Prozessterm Präfix eines Prozesses P ist, kann ohne weitere Vorbedingungen ausgeführt werden.
- Der Folgeprozess ist der restliche Prozess P.
- Diese Regel ist ein *Axiom* (ohne Prämisse).

Auswahlregel

$$[\text{choice}]\ \frac{P_j \xrightarrow{\alpha} P_j'}{\sum_{i \in I} P_i \xrightarrow{\alpha} P_j'}\ j \in I$$

- **Prämisse:** Prozess P_j kann eine Aktion α ausführen und dadurch zum Folgeprozess P_j' werden.
- **Conclusio:** Wenn P_j Teilprozess in einem Auswahlterm ist, dann ist P_j' möglicher Folgeprozess des Auswahlterms.

Ersetzungsregel

$$[\text{Rec}]\ \frac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'}\ K \coloneqq P$$

- **Prämisse:** Prozess P kann eine Aktion α ausführen und dadurch zum Folgeprozess P' werden.
- **Conclusio:** Wenn das Verhalten des Prozessnamen K durch den Prozessterm P definiert ist, dann ist P' möglicher Folgeprozess von K.

Nebenläufigkeitsregel - Linke Seite

$$[\text{Par1}] \quad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

- **Prämisse:** Prozess P kann eine Aktion α ausführen und dadurch zum Folgeprozess P' werden.
- **Conclusio:** Wenn P Teilprozess einer Parallelkomposition ist, kann P unabhängig von anderen Teilprozessen Aktion α ausführen und zum Folgeprozess P' werden.

Nebenläufigkeitsregel – Rechte Seite

$$[\text{Par2}] \quad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

- **Prämisse:** Prozess Q kann eine Aktion α ausführen und dadurch zum Folgeprozess Q' werden.
- **Conclusio:** Wenn Q Teilprozess einer Parallelkomposition ist, kann Q unabhängig von anderen Teilprozessen Aktion α ausführen und zum Folgeprozess Q' werden.

Synchronisationsregel

$$[\text{Sync}]\ \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

- **Prämisse:** Prozess P kann Aktion a ausführen und zum Folgeprozess P' werden und Prozess Q kann die zu a komplementäre Aktion $\bar{a}$ ausführen und zu Folgeprozess Q' werden.
- **Conclusio:** Wenn P und Q Teilprozesse einer Parallelkomposition sind, können beide Prozesse gemeinsam Aktion a bzw. $\bar{a}$ als τ ausführen und zu den nebenläufigen Folgeprozessen P' und Q' werden.

Umbenennungsregel

$$[\text{Ren}] \quad \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

- **Prämisse:** Prozess P kann eine Aktion α ausführen und dadurch zum Folgeprozess P' werden.
- **Conclusio:** Wird eine Umbenennungsfunktion f auf P angwendet, dann führt P die umbenannte Aktion $\mathrm{f}(\alpha)$ aus und wird zum Folgeprozess P', auf der weiterhin die Umbenennungsfunktion angewendet wird.

Restriktionsregel

$$[\text{Res}] \quad \frac{P \xrightarrow{\alpha} P'}{P\backslash \mathrm{L} \xrightarrow{\alpha} P'\backslash \mathrm{L}} \; \alpha, \bar{\alpha} \notin L$$

- **Prämisse:** Prozess P kann eine Aktion α ausführen und dadurch zum Folgeprozess P' werden.
- **Conclusio:** Wird P ohne die Restriktionsmenge L ausgeführt, dann kann P die Aktion α nur ausführen und zum Folgeprozess P' werden, falls α bzw. dessen Komplement nicht in der Menge L ist.

SOS-Regeln

- Diese Menge der SOS-Regeln spezifiziert vollständig die operationelle Semantik von CCS-Prozesstermen.
- Das Verhalten eines Prozessterms P kann durch wiederholte Anwendung der Schlussregeln konstruiert werden.

Aufgabe: Anwendung von SOS-Regeln

Es soll gezeigt werden, dass Bob nach genau einer Kaffeepause die ST-2-Hausaufgabe abgeben könnte.

$$\text{Bob} := \overline{2€} \,.\, \text{Kaffee} \,.\, \text{Bob} + \overline{ST2HA} \,.\, \mathbf{0}$$

Lösung

$$\dfrac{\dfrac{}{\overline{2€}\ .\ \text{Kaffee}\ .\ \text{Bob} \xrightarrow{\overline{2€}} \text{Kaffee}\ .\ \text{Bob}}\ [\text{pre}]}{\overline{2€}\ .\ \text{Kaffee}\ .\ \text{Bob} + \overline{ST2HA}\ .\ \mathbf{0} \xrightarrow{\overline{2€}} \text{Kaffee}\ .\ \text{Bob} \xrightarrow{\text{Kaffee}} \text{Bob}}\ [\text{choice}] \quad [\text{pre}] \quad \textbf{(1)}$$

$$\dfrac{\dfrac{\dfrac{}{\overline{ST2HA}\ .\ \mathbf{0} \xrightarrow{\overline{ST2HA}} \mathbf{0}}\ [\text{pre}]}{\overline{2€}\ .\ \text{Kaffee}\ .\ \text{Bob} + \overline{ST2HA}\ .\ \mathbf{0} \xrightarrow{\overline{ST2HA}} \mathbf{0}}\ [\text{choice}]}{\text{Bob} \xrightarrow{\overline{ST2HA}} \mathbf{0}}\ [\text{rec}] \quad \text{Bob} := \overline{2€}\ .\ \text{Kaffee}\ .\ \text{Bob} + \overline{ST2HA}\ .\ \mathbf{0} \quad \textbf{(2)}$$

(1) + **(2)** q.e.d.

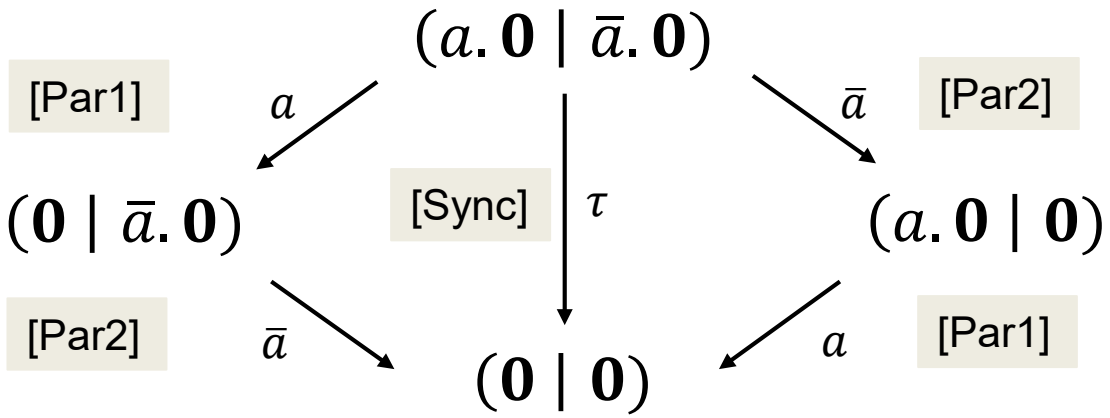
Anmerkungen zur Lösung

- Derartige Sequenzen von Schlussregelanwendungen auf Prozesstermen zum Beweis der Erreichbarkeit anderer Prozessterme über Aktionsfolgen werden **Ableitungsbäume** genannt.
- Im Beispiel könnte man (2) noch als „rechten Ast“ neben (1) anbringen um einen Baum zu erhalten (aus Platzmangel hier nicht dargestellt).

Synchronisation durch Restriktion

Abschließend betrachten wir noch einmal das Zusammenspiel von Synchronisation und Restriktion.

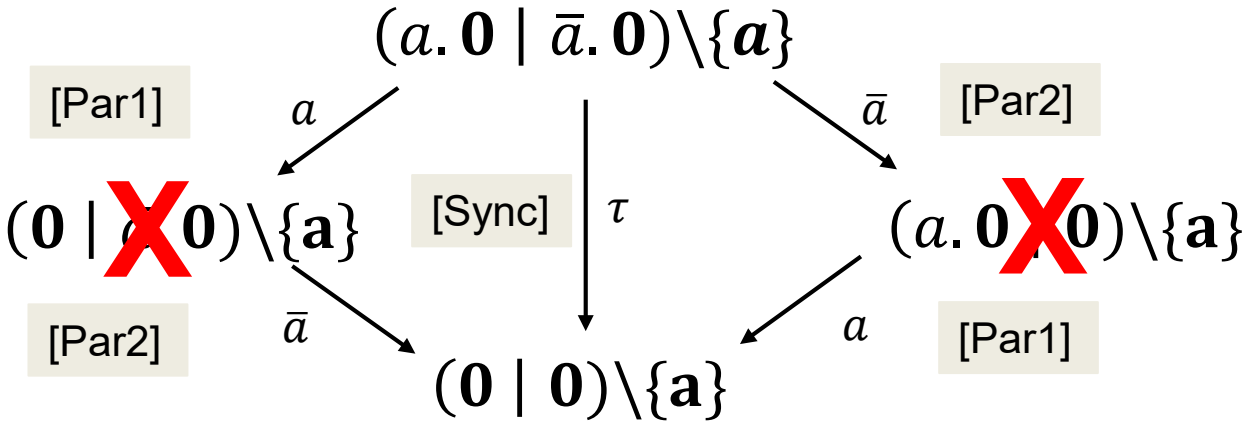
Beispiel ohne Restriktion:



Synchronisation durch Restriktion

Abschließend betrachten wir noch einmal das Zusammenspiel von Synchronisation und Restriktion.

Beispiel mit Restriktion:



Synchronisation durch Restriktion

- Der Restriktionsoperator verbietet Schritte, die sichtbar mit Aktion a bzw. $\bar{a}$ beschriftet sind.
- Die beiden Teilprozesse können nur fortsetzen, in dem sie sich synchronisieren und somit a bzw. $\bar{a}$ zu τ werden lassen (das durch den Restriktions-operator stets „durchgelassen“ wird).
- Auf diese Weise kann Prozesssynchronisation für bestimmte Aktionen innerhalb eines bestimmten Scopes in CCS erzwungen werden.

Kongruenzregeln für CCS

Motivation

- Für die Semantik des Kompositionsoperators „|“ gibt es insgesamt drei SOS-Regeln, wobei die beiden Regeln *[Par1]* und *[Par2]* symmetrisch sind.
- Intuitiv würden wir auch erwarten, dass der Kompositionsoperator symmetrisch ist:

$$P \mid Q \stackrel{?}{=} Q \mid P$$

- Aber: Die Verwendung des Gleichheitszeichens ist hier definitiv falsch, da beide Prozessterme **syntaktisch unterschiedlich** sind.

Syntaktische Kongruenz

- Statt syntaktischer Gleichheit sprechen wir von Kongruenz zweier Prozessterme:

$$P \equiv Q$$

  wenn beide Prozesse syntaktisch verschieden sein könnten, aber *semantisch äquivalent* hinsichtlich der LTS-Semantik sind.
- Problem: Zur vollständigen Identifikation aller semantisch äquivalenten Terme müsste das (nicht-endliche) LTS für ganz $\mathcal{P}$ konstruiert werden (dazu später mehr).
- Stattdessen wollen wir zunächst rein syntaktische Kongruenzregeln auf der Termstruktur betrachten.

Kongruenzregeln für CCS-Prozessterme

(1) $P + \mathbf{0} \equiv P$

(2) $P \mid \mathbf{0} \equiv P$

(3) $P + Q \equiv Q + P$

(4) $P \mid Q \equiv Q \mid P$

(5) $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$

(6) $P + (Q + R) \equiv (P + Q) + R$

(7) $P\backslash \mathrm{L}\backslash \mathrm{L}' \equiv P(L \cup L')$

(8) $P[f][f'] \equiv P[f \circ f']$

Anmerkungen zu den Kongruenzregeln

- Die Regeln (1) und (2) besagen, dass $\mathbf{0}$ eine Art „neutrales Element“ hinsichtlich „+“ und „|“ ist.
- Die Regeln (3) und (4) besagen, dass „+“ und „|“ kommutativ sind.
- Die Regeln (5) und (6) besagen, dass „+“ und „|“ assoziativ sind.
- Regel (7) besagt, dass mehrere Restriktionen hintereinander zu einer Restriktion vereinigt werden können.
- Regel (8) besagt, dass mehrere Umbenennungen hintereinander zu einer Umbenennung kombiniert werden können (via Funktionskomposition $\circ$).

Weitere Kongruenzregeln

- $P + P \equiv \mathrm{P}$
  Kann weggelassen werden, falls angenommen wird, dass Indizes eindeutig Prozessterme identifizieren.
- $K \equiv P$
  falls $K \coloneqq P$
  offensichtlich.
- $P \backslash \mathrm{L} \equiv P$
  falls $L(P) \cap (L \cup \bar{L}) = \emptyset$
  wobei $L(P)$ die Menge der Labels in Term P bezeichnet.
- $P[f] \equiv P$
  falls $f = id$
  wobei id die Identitätsfunktion bezeichnet.

Einschub: Äquivalenzrelation

Die Kongruenzrelation $\equiv \subseteq \mathcal{P} \times \mathcal{P}$ ist eine **Äquivalenzrelation** auf der Menge aller CCS-Prozessterme:

- Reflexivität: $\forall P \in \mathcal{P}: P \equiv P$
- Symmetrie: $\forall P, Q \in \mathcal{P}: P \equiv Q \Leftrightarrow Q \equiv P$
- Transitivität: $\forall P, Q, R \in \mathcal{P}: \mathrm{P} \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$

Mit $[P]/_{\equiv} = \{Q \mid Q \in \mathcal{P} \wedge Q \equiv P\}$ bezeichnen wir die **Äquivalenzklasse** des Prozessterms $P \in \mathcal{P}$. Die Menge aller Äquivalenzklassen ist eine **Partition** von $\mathcal{P}$.

Falsche Kongruenzregeln

X $\alpha.(P + Q) \equiv \alpha.P + \alpha.Q$

Distributivgesetz zwischen „.“ und „+“ gilt nicht.

X $\tau.P \equiv P$

Nicht sichtbare Aktionen können nicht einfach weggelassen werden.

Begründungen folgen…

UNIVERSITÄT SIEGEN

Äquivalenzbegriffe für LTS

Motivation

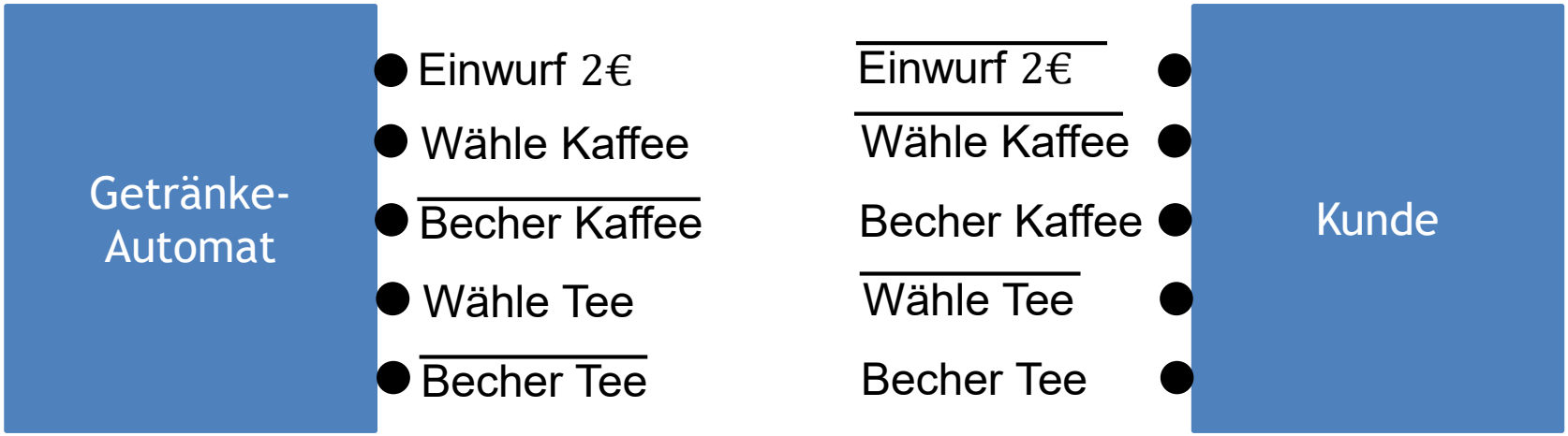
Für zwei Prozesse P und P' kann nachgewiesen werden, dass sie semantisch äquivalent sind, indem versucht wird, durch Anwendung von Kongruenzregeln $P \equiv P'$ nachzuweisen.

1. **Problem:** Je nach Größe der Terme sind dafür eventuell sehr viele Beweisschritte notwendig.
2. **Problem:** Ein vollständig automatisiertes Beweisverfahren ist nicht möglich.
3. **Problem:** Eine (endliche) Menge syntaktischer Kongruenzregeln kann nie vollständig semantische Äquivalenz charakterisieren.

Semantische Äquivalenzbegriffe

- Für zwei Prozessterme P und P' wollen wir nun Äquivalenzkriterien betrachten, die nicht auf Basis von Termstrukturen definiert sind, sondern auf Basis der LTS-Semantik $(\mathcal{P}, Act, \rightarrow)$.
- Dazu sollen entsprechende Vergleichskriterien für Zustände $P \in \mathcal{P}$ und $P' \in \mathcal{P}$ hinsichtlich ihres Folgeverhaltens gemäß der Transitionsrelation $\rightarrow$ definiert werden.

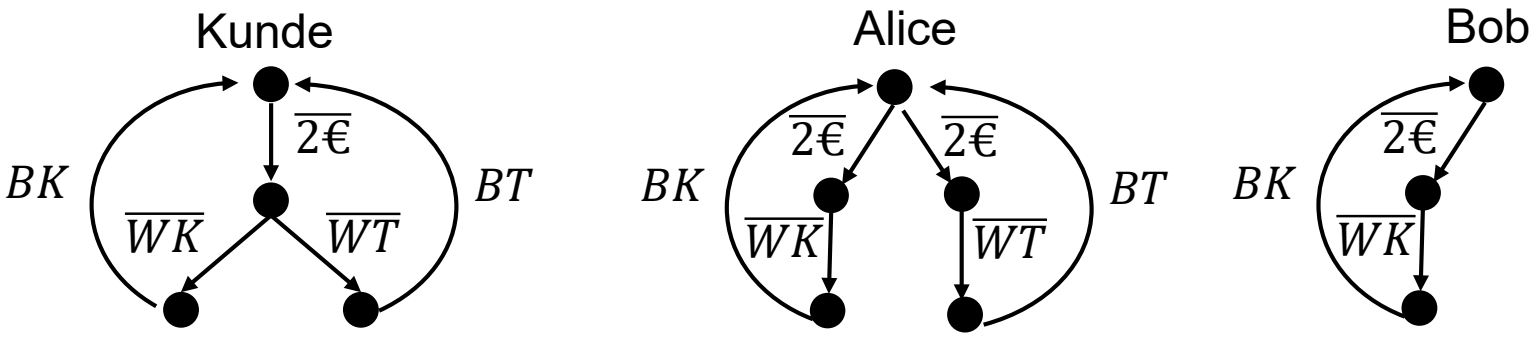
Aufgabe: Kompatibilität von Prozessen



Wie muss sich ein Kunde verhalten, um den Getränkeautomaten richtig zu bedienen?

(Wir verwenden nachfolgend die Abkürzungen 2€, WK, BK, WT, BT)

Aufgabe: Kompatibilität von Prozessen



- Der Spezifikation des Automaten-Herstellers für das gewünschte Kundenverhalten liegt als CCS-Prozess (hier: LTS) vor.
- Das Kundenverhalten von Alice und Bob liegt ebenfalls jeweils als CCS-Prozess vor.
- Worin unterscheidet sich das Verhalten von Alice und Bob vom gewünschten Kundenverhalten? Stellt das ein Problem dar?

Einschub: Initialzustände

Zur Erinnerung:

- Die Definition eines LTS beinhaltet *keinen* expliziten Initialzustand.
- Die operationelle Semantik von CCS wird durch *ein* LTS angegeben.

In den folgenden Beispielen weichen wir davon ab:

- Eigentlich sind die drei LTS auf der vorherigen Folien Teilgraphen eines einzigen großen LTS.
- Der Zustand, der mit dem Namen des jeweiligen Prozesses (Kunde, Alice, Bob) beschriftet ist, stellt eine Art Initialzustand dar. Alternativ heißt dieser Zustand auch häufig s_0 bzw. P_0.
- Auf die Benennung anderer Zustände verzichten hingegen zumeist.

(Zwischen-)Lösung

- Das gewünschte Kundenverhalten sieht vor, dass der Kunde zunächst ein Geldstück einwirft und sich dann zwischen Kaffee und Tee entscheidet.
- Alice trinkt gern Kaffee und Tee, aber sie hat sich immer schon festgelegt, bevor sie das Geldstück einwirft.
- Bob hingegen trinkt nur Kaffee, Tee kommt für ihn nicht in Frage.

Vergleich von Spurenmengen

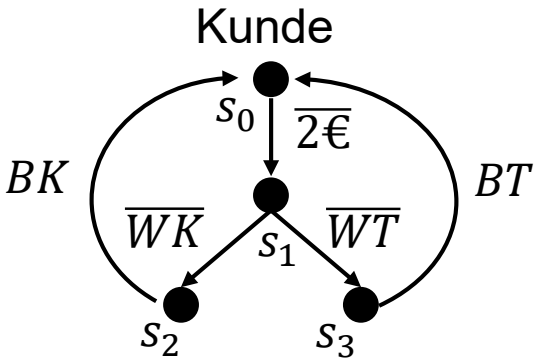
- Das gewünschte Kundenverhalten unterscheidet sich vom Verhalten von Alice und Bob hinsichtlich der Struktur der LTS-Graphen.
- Die Menge möglicher Aktionssequenzen des gewünschten Kundenverhaltens von Alice ist hingegen gleich (bzw. im Falle von Bob eine Teilmenge).
- Die Aktionssequenzen bezeichnet man als **Spuren** (engl. **Traces**).

Definition: Spurmenge

Die **Spurmenge** $tr(P) \subseteq Act^*$ eines Prozesses $P \in \mathcal{P}$ ist induktiv definiert als die maximale Menge von Aktionssequenzen, die folgende Regeln erfüllt:

1. $\epsilon \in tr(P)$
2. $\alpha\omega \in tr(P)$ falls $P \xrightarrow{\alpha} P'$ und $\omega \in tr(P')$

Beispiel: Spurenmenge



Benennung der Zustände für die Zuordnung der Spurenmengen.

$$tr(s_0) = \{\epsilon\} \cup \{\overline{2€} \cdot \omega_1 \mid \omega_1 \in tr(s_1)\}$$

$$tr(s_1) = \{\epsilon\} \cup \{\overline{WK} \cdot \omega_2 \mid \omega_2 \in tr(s_1)\} \cup \{\overline{WT} \cdot \omega_3 \mid \omega_3 \in tr(s_3)\}$$

$$tr(s_2) = \{\epsilon\} \cup \{BK \cdot \omega_0 \mid \omega_0 \in tr(s_0)\}$$

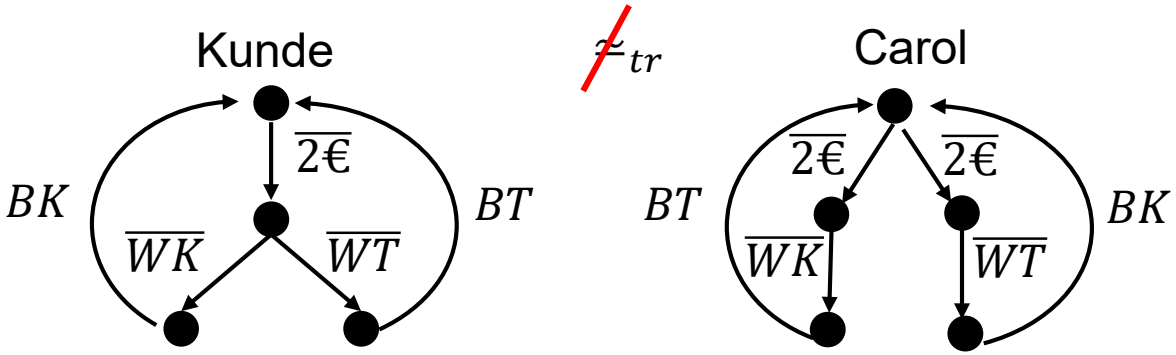
$$tr(s_3) = \{\epsilon\} \cup \{BT \cdot \omega_0 \mid \omega_0 \in tr(s_0)\}$$

$$tr(s_0) = \{\epsilon, \overline{2€}, \overline{2€} \cdot \overline{WK}, \overline{2€} \cdot \overline{WT}, \overline{2€} \cdot \overline{WK} \cdot BK, \overline{2€} \cdot \overline{WT} \cdot BT, \overline{2€} \cdot \overline{WK} \cdot BK \cdot \overline{2€}, \overline{2€} \cdot \overline{WT} \cdot BT \cdot \overline{2€}, \dots\}$$

Definition: Spuräquivalenz

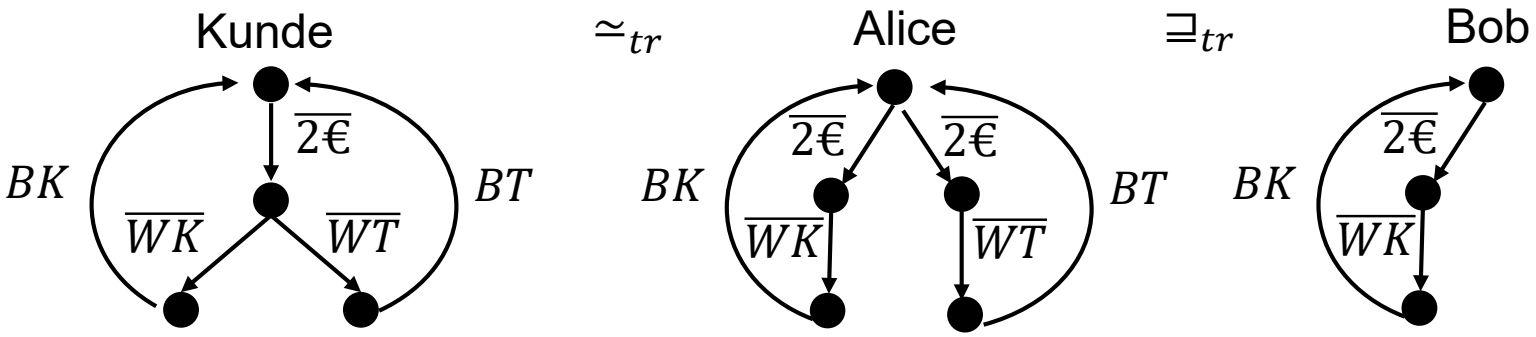
- Zwei Prozesse P und P' sind **spuräquivalent**, falls $tr(P) = tr(P')$ gilt.
- Mit $\simeq_{tr} \subseteq \mathcal{P} \times \mathcal{P}$ bezeichnen wir die **Spuräquivalenzrelation** auf der Menge aller CCS-Prozessterme.
- Zwischen Prozess P und P' gilt **Spurinklusion**, falls $tr(P) \subseteq tr(P')$ gilt.
- Mit $\sqsubseteq_{tr} \subseteq \mathcal{P} \times \mathcal{P}$ bezeichnen wir die **Spur-inklusions-Quasiordnung** auf der Menge aller CCS-Prozessterme.

Beispiel: Spuräquivalenz



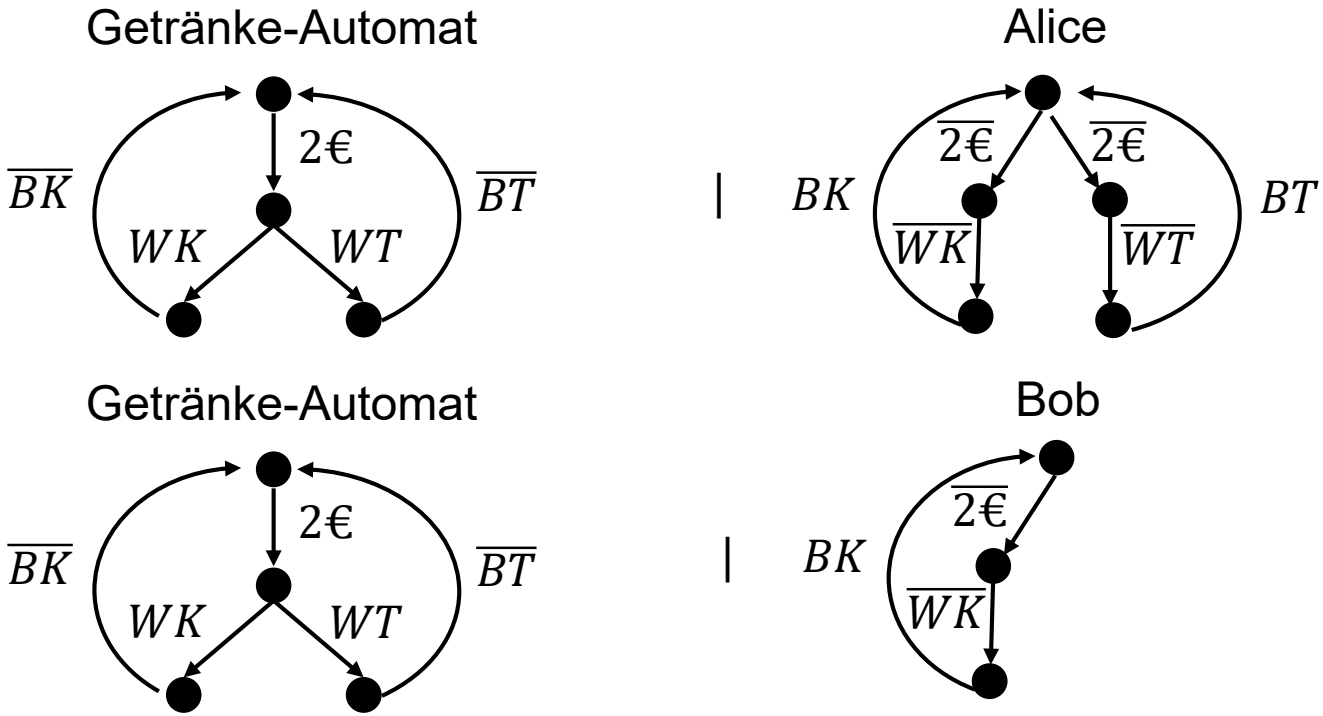
- Carol‘s Verhalten (Auswahl eines Kaffees, aber Erwarten eines Tees und umgekehrt) ist nicht spuräquivalent zum gewünschten Kundenverhalten und wird somit abgelehnt.
- Die Forderung nach Spuräquivalenz würde also solchen unsinnigen Aktionssequenzen verhindern.

Beispiel: Spuräquivalenz



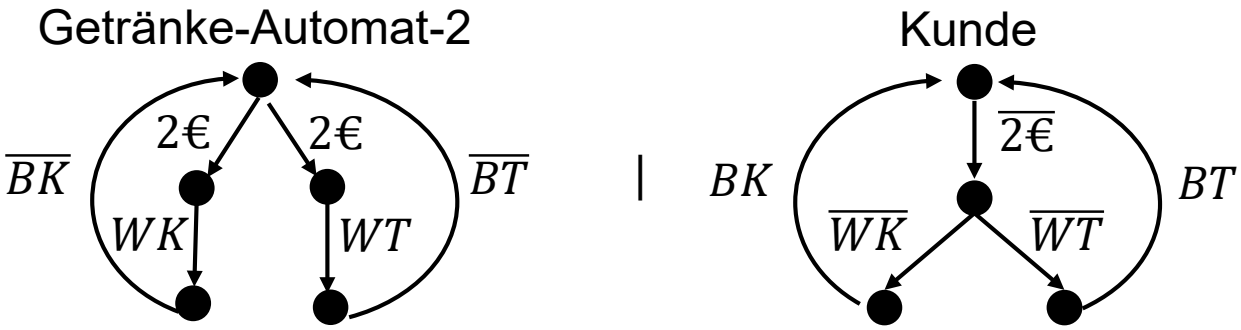
- Alice‘s Verhalten ist spuräquivalent zum gewünschten Kundenverhalten.
- Bob‘s Verhalten ist spurinkludiert im gewünschten Kundenverhalten.
- Fazit: Beide können gefahrlos den Automaten nutzen? Um das zu beantworten, müssen wir in den Automaten schauen.

Beispiel: Kompatibilität von Prozessen



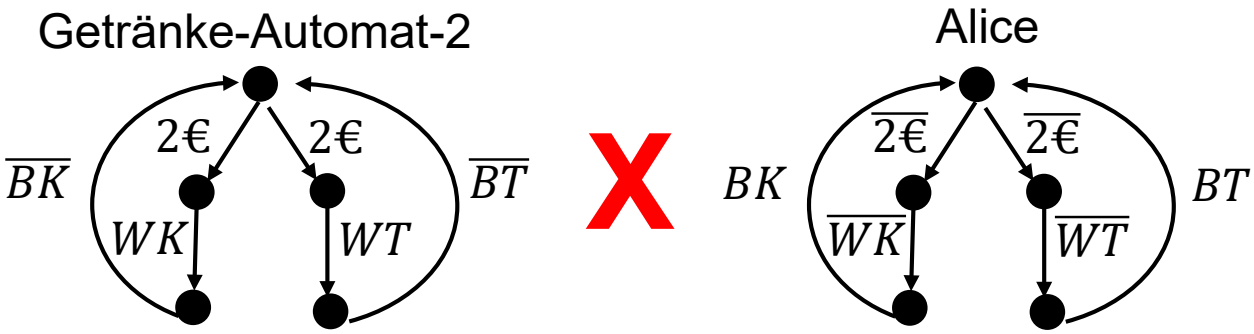
Dieser Getränke-Automat kann wie gewünscht mit Alice und Bob interagieren.

Beispiel: Kompatibilität von Prozessen



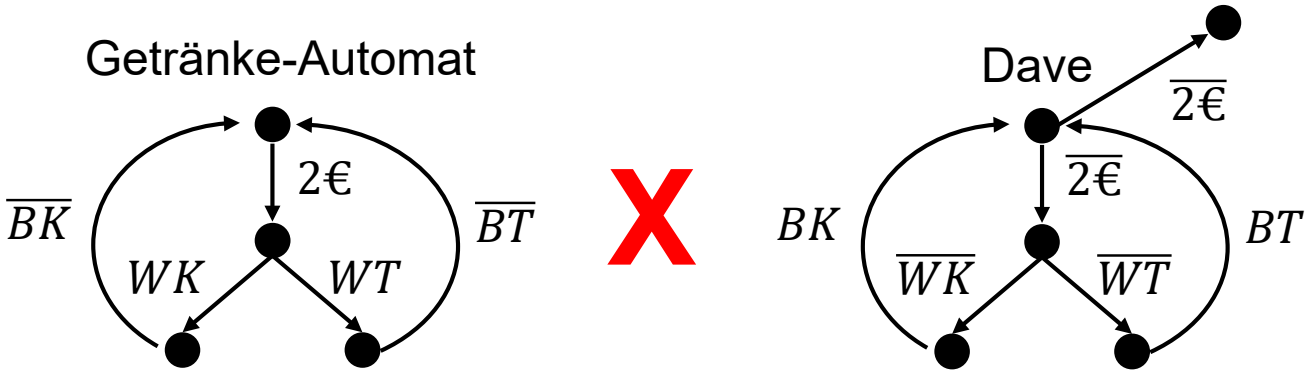
- Dieser alternative „Getränke-Automat-2“ wäre ebenfalls kompatibel zum gewünschten Kundenverhalten.
- … und damit auch zu Alice‘s und Bob‘s Verhalten?

Beispiel: Kompatibilität von Prozessen



- Alice kann die Getränke-Auswahl nach dem Geldeinwurf nicht mehr beeinflussen.
- Wenn sich der Automat und Alice nach dem Geldeinwurf für jeweils unterschiedliche Getränke entscheiden, bleibt der Ablauf für immer stecken (**Deadlock**).
- Das gleiche Problem besteht auch bei Bob‘s Verhalten.

Beispiel: Kompatibilität von Prozessen

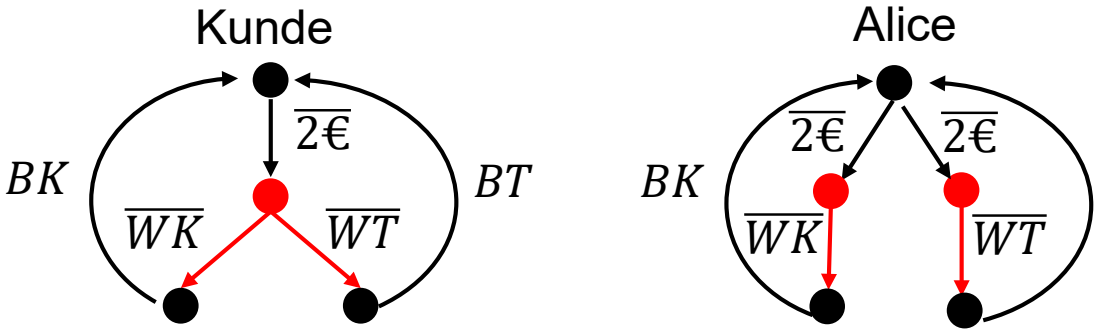


- Dave‘s Verhalten ist ebenfalls spuräquivalent zum gewünschten Kundenverhalten, obwohl er gegebenenfalls nach dem Geldeinwurf die Interaktion einfach beendet.
- Weiteres Problem der Spuräquivalenz: keine Unterscheidung zwischen beendeten und nicht beendeten Spuren.

Diskussion: Spuräquivalenz

- Spuräquivalenz ist ein naheliegender Äquivalenzbegriff, der aber zu schwach ist, sobald die betrachteten Prozesse potentiell nichtdeterministisches Verhalten aufweisen.
- In diesem Fall können Prozesse in der Interaktion mit anderen Prozessen unterschiedliches Deadlock-Verhalten aufweisen, obwohl sie spuräquivalent sind.
- Problem: Spuräquivalenz ist insensitiv hinsichtlich der **Verzweigungsstruktur** und des **Terminierungs-verhaltens** nichtdeterministischer Prozesse.

Simulation von Zuständen



- Zu Beginn können der Musterkunde und Alice das Gleiche tun: Geld einwerfen.
- Der Musterkunde erreicht nach dem Geldeinwurf *einen* möglichen Folgezustand, in dem noch beide Auswahlaktionen möglich sind.
- Alice erreicht hingegen nach dem Geldeinwurf einen von *zwei* möglichen Folgezuständen, in dem jeweils nur noch eine Auswahlaktion möglich ist.
- Wir sagen dazu: Keiner dieser beiden Zustände kann das Folgeverhalten des entsprechenden Zustandes des Musterkunden vollständig **simulieren**.

Definition: Bisimulation

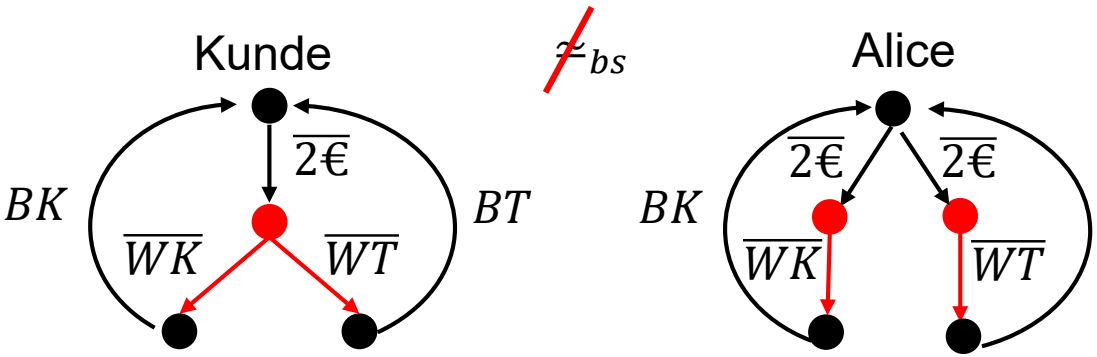
Eine Relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ ist eine **Bisimulation**, genau dann wenn für alle $(P_1, P_2) \in \mathcal{R}$ und alle $\alpha \in Act$ gilt:

- Wenn $P_1 \xrightarrow{\alpha} P_1'$, dann $P_2 \xrightarrow{\alpha} P_2'$ so dass $(P_1', P_2') \in \mathcal{R}$
- Wenn $P_2 \xrightarrow{\alpha} P_2'$, dann $P_1 \xrightarrow{\alpha} P_1'$ so dass $(P_1', P_2') \in \mathcal{R}$

Bisimilarität und Bisimulationsäquivalenz

- Zwei Prozesse P und P' sind **bisimilar**, wenn eine Bisimulation $\mathcal{R}$ existiert sodass $(P, P') \in \mathcal{R}$.
- Mit $\simeq_{bs} \subseteq \mathcal{P} \times \mathcal{P}$ bezeichnen wir die **Bisimulationsäquivalenzrelation** auf der Menge aller CCS-Prozessterme.
- Es gibt hier auch wieder eine entsprechende Quasiordnung $\sqsubseteq_s$, die nur in eine Richtung gilt (Simulationsrelation).

Beispiel: Bisimulation



- Es lässt sich keine Bisimulationsrelation angeben, die dem roten Zustand von „Kunde“ *einen* ihn simulierenden Zustand von „Alice“ zuordnet.
- Beide Prozesse sind somit nicht bisimilar.

Bisimulationsäquivalenz und Spuräquivalenz

Satz. $\simeq_{bs} \subseteq \simeq_{tr}$

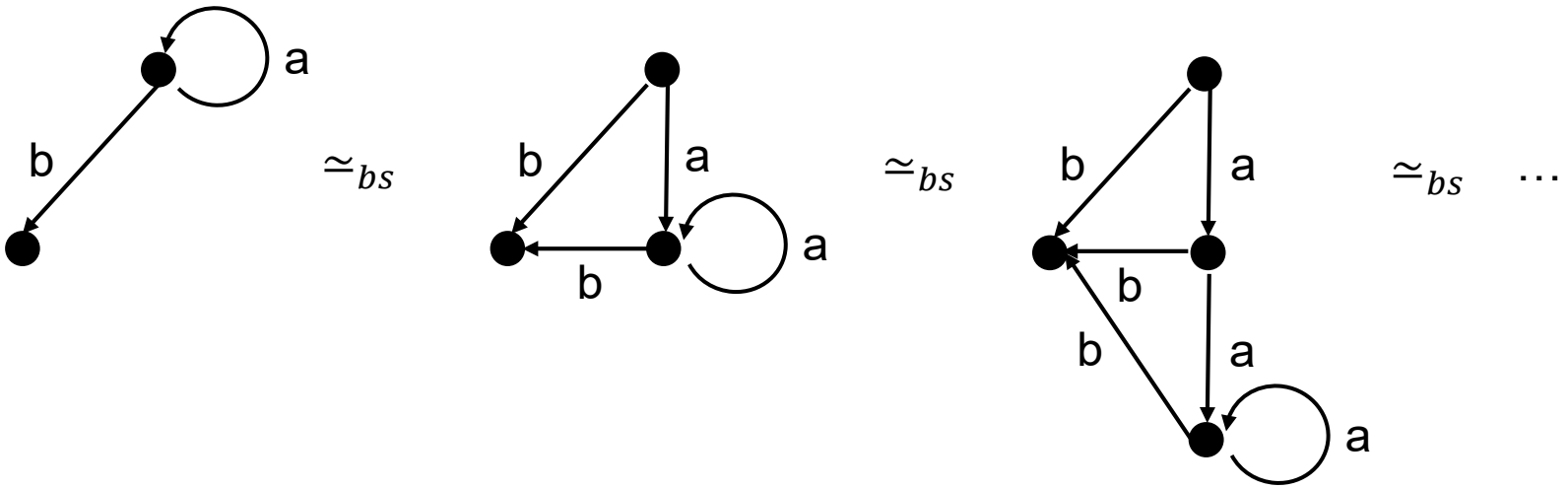
- Mit anderen Worten: Sind zwei Prozesse bisimilar, dann sind sie auch spuräquivalent, aber nicht umgekehrt.
- Bisimulation ist ein strengeres Äquivalenzkriterium als Spuräquivalenz.

Bisimulationsäquivalenz und Kongruenz

Satz. $\equiv \subseteq \simeq_{bs}$

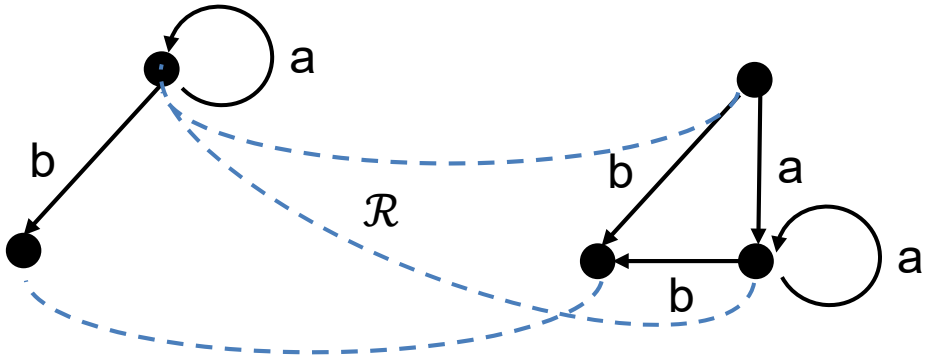
- Kongruenz ist somit (u. U. unnötig) strenger als Bisimulationsäquivalenz.
- Die Umkehrung gilt im Allgemeinen nicht, da Bisimulationsäquivalenz „viel genauer hingucken kann“, ob zwei Prozesse das gleiche Verhalten haben.
- Aber: Welche Prozessvariationen lässt Bisimulation denn überhaupt zu?

Was lässt Bisimulation zu?



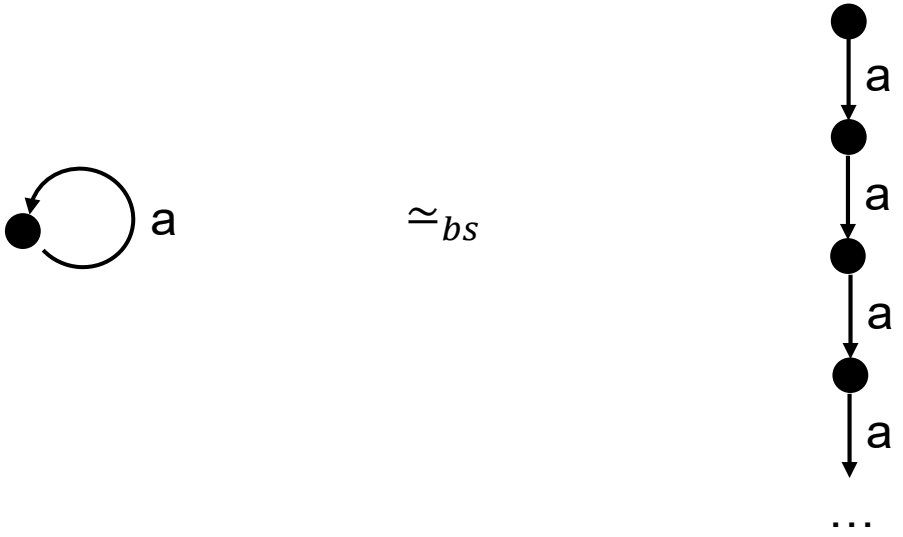
- Keine Unterscheidung zwischen verschieden tief **„abgerollten“ Schleifen**.

Beispiel: Bisimulation



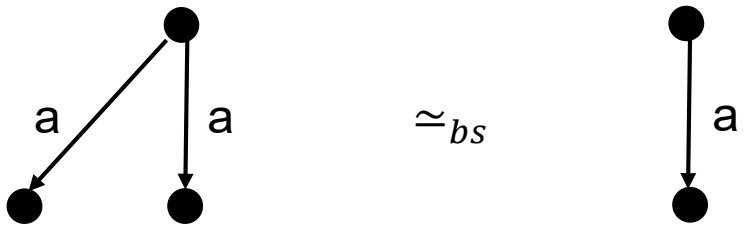
- Bisimulation $\mathcal{R}$ als „Zeuge“.
- Beobachtung: ein Zustand des einen LTS kann mehrere Zustände des anderen LTS simulieren.

Was lässt Bisimulation zu?



- Keine Unterscheidung zwischen **Konvergenz** (Festlaufen in einem Zustand) und **Divergenz** (Durchlaufen einer unendlichen Zustandsfolge).
- Spezialfall des Schleifenabrollens: unendliches Abrollen.

Was lässt Bisimulation zu?



- Keine Unterscheidung zwischen deterministischem Verhalten und **unbeobachtbarem nichtdeterministischem** Verhalten**.**
- Erlaubt redundante Teilkomponenten im LTS.

Bisimulationsäquivalenz und CCS-Operatoren

Satz. Aus $P \simeq_{bs} P'$ folgt
(1) $\alpha.P \simeq_{bs} \alpha.P'$
(2) $P + Q \simeq_{bs} P' + Q$
(3) $P \mid Q \simeq_{bs} P' \mid Q$

- Die Relation $\simeq_{bs}$ heißt dann *Kongruenz* bezüglich der Operatoren „.“, „+“ und „|“ (auch für „\“ und „[…]“)
- Dieser Satz garantiert, dass der Austausch von Teilprozessen durch dazu bisimilare Teilprozesse das Verhalten des Gesamtsystems nicht verändert.

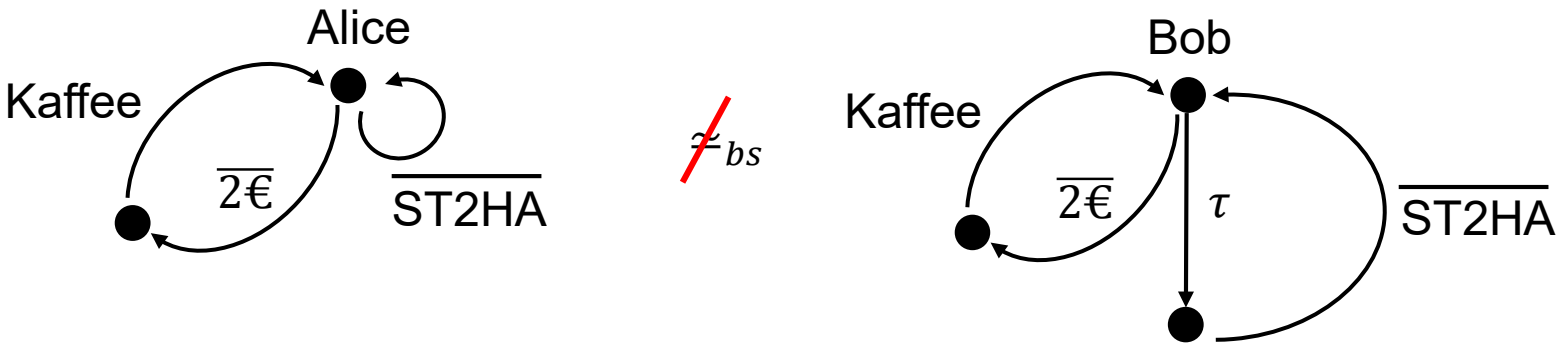
Beispiel: Ersetzung von Prozessen

- Eigenschaft (3) garantiert nun, dass die Ersetzung des „Musterkunden“ durch einen bisimilaren echten Kunden korrekt mit dem Getränkeautomaten interagieren wird.
- „Korrekt“ heißt hier, das der entstehende Interaktionsprozess bisimilar zum idealen Interaktionsprozess zwischen dem Musterkunden und dem Getränkeautomaten ist.

Kontrollierbare und Beobachtbare Aktionen

- **Eingabeaktionen** a sind von außen kontrollierbare, innerhalb des Prozesses beobachtbare Aktionen. Aus Sicht des Prozesses treten Eingaben immer „nichtdeterministisch“ (sporadisch) in der Umgebung auf.
- **Ausgabeaktionen** $\bar{a}$ sind von außen beobachtbare, innerhalb des Prozesses kontrollierbare Aktionen. Im Prozess kann es je nach Zustand eine nichtdeterministische Auswahl zwischen Ausgabeaktionen geben.
- τ-steps sind **interne (stille, versteckte) Aktionen**, die von außen weder beobachtbar, noch kontrollierbar sind und nichtdeterministisch auftreten.

Beispiel: τ-Schritte

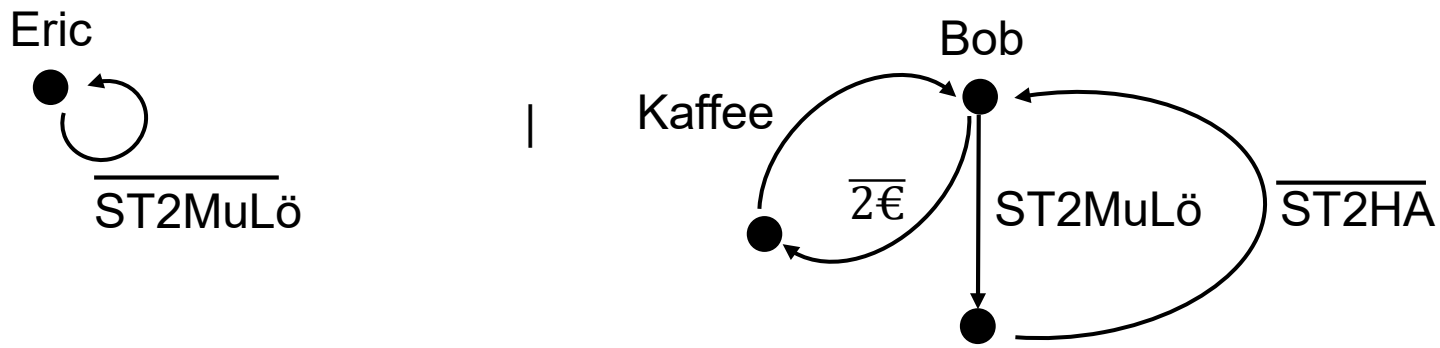


- Alice macht immer mal wieder Kaffeepausen und geht irgendwann die nächste ST-2-Hausaufgabe an.
- Bob macht auch immer mal wieder Kaffeepausen, aber irgendwann treibt ihn das schlechte Gewissen per τ-Schritt in einen Zustand, in dem er sich keine Kaffeepausen mehr gönnt, bis die nächste ST-2-Hausaufgabe endlich fertig ist.
- Die beiden Prozesse sind nicht bisimilar, da Alice den τ-Schritt nicht simulieren kann. Ist das sinnvoll?

Starke und Schwache Bisimulation

- Die bisher definierte Bisimulation wird **starke Bisimulation** (strong bisimulation) genannt.
- **Schwache Bisimulation** ignoriert hingegen τ-Schritte (wir verzichten an dieser Stelle auf eine formale Definition).
- Unter **schwacher Bisimulation** (weak bisimulation) würden sich Alice und Bob somit bisimilar verhalten.

Beispiel: Schwache und Starke Bisimulation



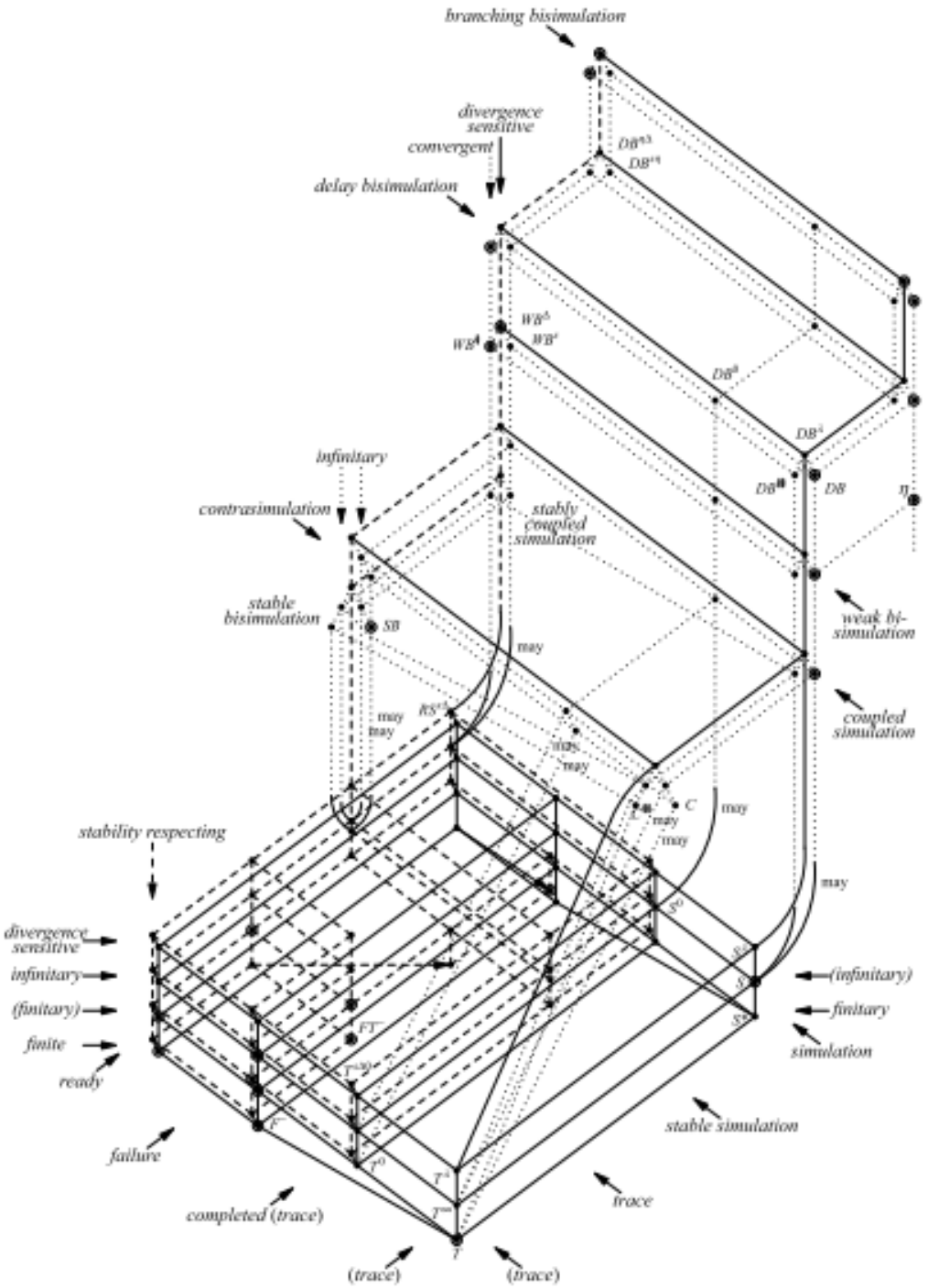
Schwache Bisimulation kann unter Umständen zu schwach sein: Das schlechte Gewissen von Bob entpuppt sich als sein Kommilitone Eric, der die ST-2-MuLö vom letzten Jahr in die Kaffeepause mitbringt…

Schwache Bisimulation ist keine Kongruenz

- Das vorherige Beispiel zeigt, dass aus gutem Grund die Kongruenzregel $\tau.P \equiv P$ falsch ist.
- Da aber beide Prozesse schwach bisimilar sind, ist die schwache Bisimulationsäquivelanz-relation keine Kongruenz hinsichtlich der CCS-Operatoren.

Äquivalenz-Spektrum

- Neben den drei betrachteten Äquivalenzdefinitionen gibt es Unmengen weiterer Äquivalenzen.
- Diese bilden ein Spektrum, dessen Dimensionen unterschiedliche Garantien für Verhaltenseigenschaften repräsentieren.



Diskussion und Ausblick

- Die Komplexität eines Bisimulations-Checks auf endlichen LTS ist davon abhängig, ob nichtdeterministisches Verhalten vorliegt.
- Alternativ können Korrektheitsanforderungen an CCS-Prozesse durch Temporallogiken spezifiziert und verifiziert werden (Grundlage hierfür ist die Hennessy-Milner-Logik).
- Es gibt sehr viele Erweiterungen und Variationen von CCS, die sich auf bestimmte Aspekte fokussieren (z.B. Value-Passing CCS für Datenfluss, π-Kalkül für mobile Prozesse etc.)

Prüfungsstoff

- Eigenschaften und die Problematik der Qualitätssicherung Reaktiver Systeme erläutern können.
- Syntax und Semantik von CCS verstehen und auf einfache Beispiele anwenden können.
- Die Bedeutung von Kongruenzregeln und Äquivalenzbegriffen verstehen und auf einfache Beispiele anwenden können.

Literatur

- R. Milner: *Communication and Concurrency*, Prentice-Hall International, Englewood Cliffs, 1989.
- R. Keller, *Formal verification of parallel programs*, Comm. ACM, 19 (1976), pp. 371–384.
- R. V. Glabbeek, *The linear time-branching time spectrum I + II*. Handbook of Process Algebra, North-Holland, Amsterdam, 2001, pp. 3–99.