Modellierung verteilter Systeme
(Grundlagen der Programm- und Systementwicklung II)

05 – Kommunizierende Prozesse

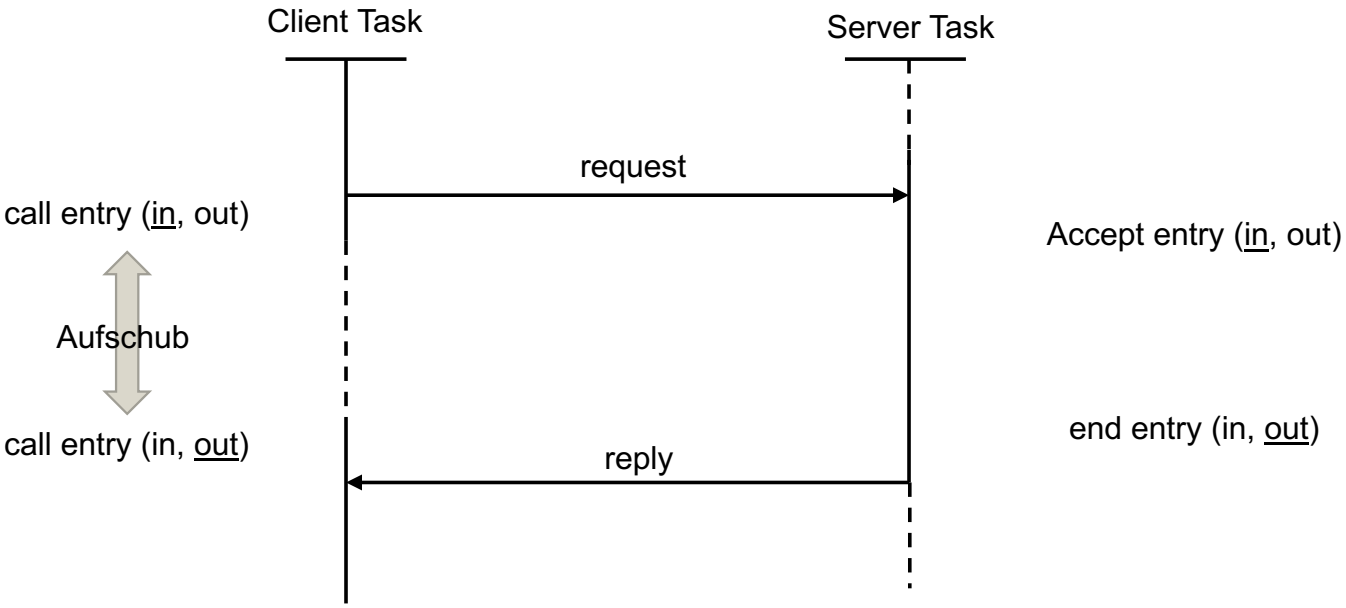
Dr. Sebastian Voss
fortiss GmbH
Kompetenzfeldleiter Model-based Systeme Engineering

fortiss

Themenübersicht

1. Einführung
2. Grundlagen: Verhalten, Interaktion, Nebenläufigkeit
3. Sequentielle Programme und Koroutinen
4. Datenflussmodelle
5. Kommunizierende Prozesse
    1. Sequentielle Prozesse
    2. Nebenläufige Prozesse
6. Zustandsübergangssysteme
7. Koordination
8. Ausführungen
9. Verhaltensspezifikationen
10. Erweiterte Themen

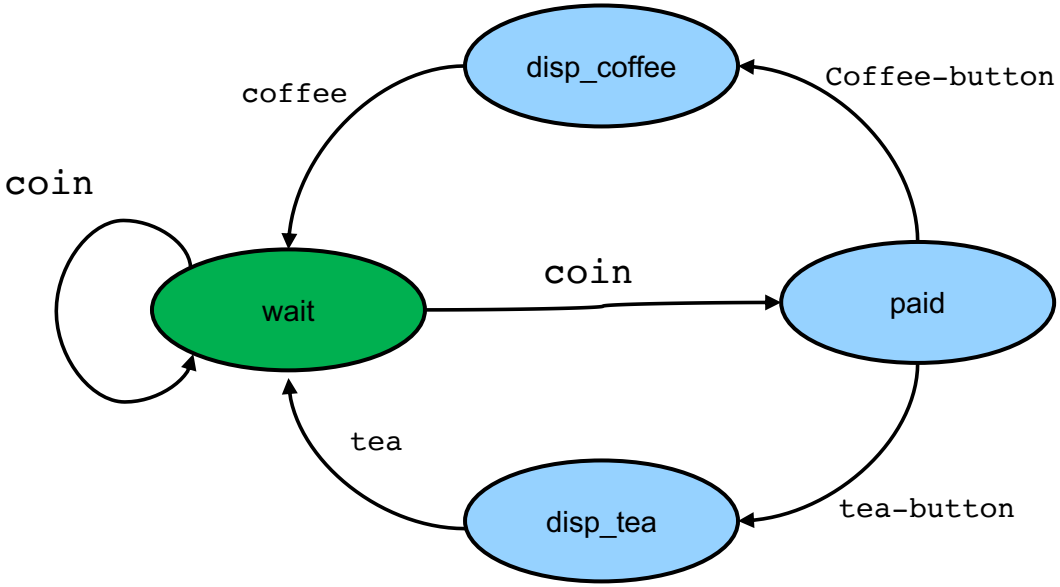
Motivation: Blockierende Interaktion



Rendezvous-Based Communication:

- Synchronous communication zwichen Sender und Empfänger
- Beispiel: Occam input/output, Ada rendezvous,

Wiederholung: Modelle reaktiver Systeme

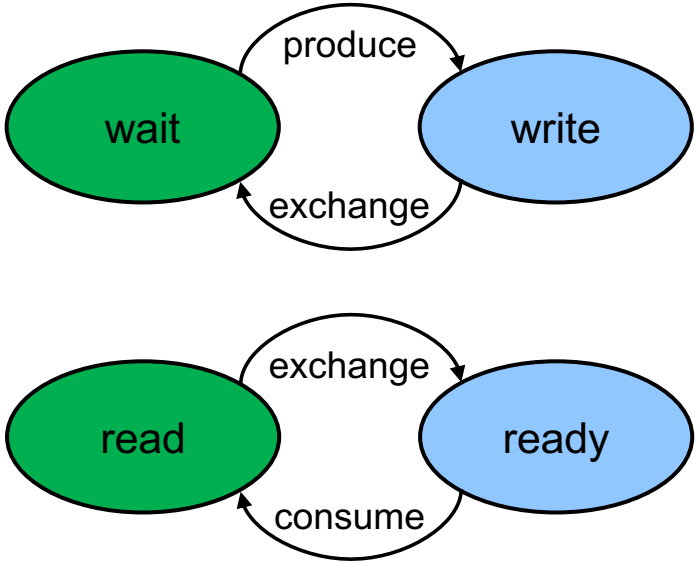


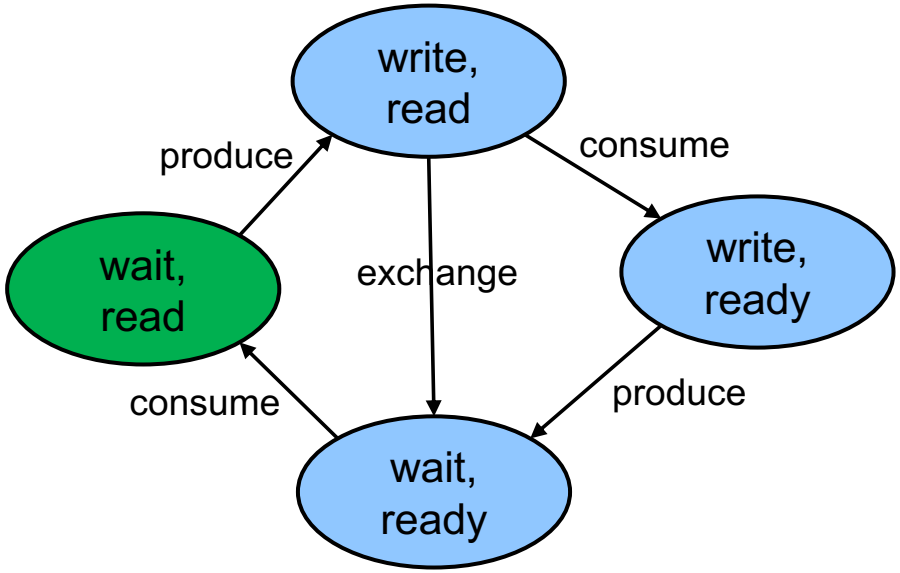
Konzepte:

- Interaktion (Alphabet): Gemeinsame Aktionen/Beobachtungen System / Umgebung
- (Beobachtbarer) Ablauf: Sequenz von Interaktionen einer Ausführung
- Auswahl: Verhaltensalternativen, die ein System anbietet
- Nichtdeterminismus: Verhaltensalternativen, die ein System erzwingt
- Eingabe/Ausgabe: Interaktion, von Umgebung/System kontrolliert

Modell: Markiertes Transitionssystem (S,A,S_0,T)

Wiederholung: Modelle der Nebenläufigkeit



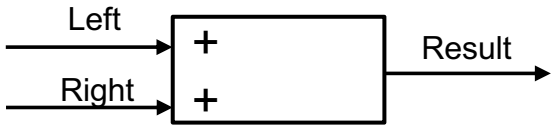


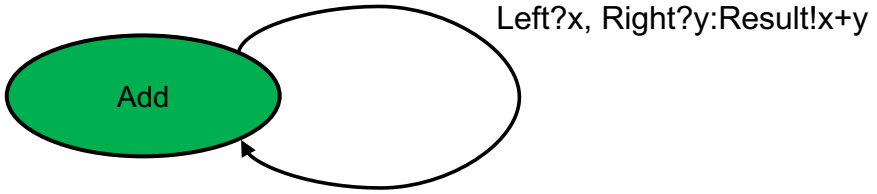
Konzepte:

- Nebenläufige Ausführung: Synchronisierte simultane Ausführung
- Unabhängige Interaktion: Wahlweise Ausführung
- Synchronisierte Interaktionen: Gemeinsame Ausführung

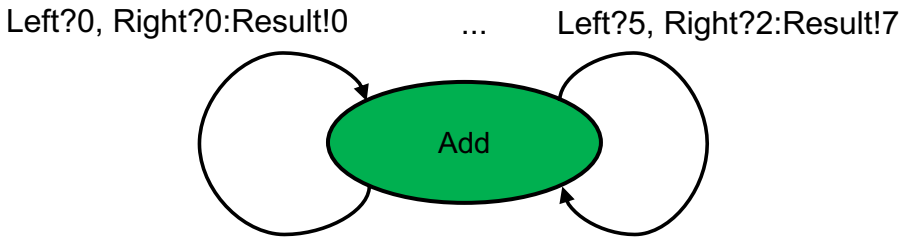
Modell: Synchronisierte Produkttransitionssysteme

Wiederholung: Modelle für Datenfluss





Left	Right	Result
x	y	x+y



Konzepte:

- Asynchroner Datenfluss: Nichtblockierende Interaktion
  - Sender kann Signal/Nachricht immer bereitstellen
  - Empfänger kann verfügbares Signal/Nachricht nach Wahl entgegennehmen (Verlust oder Pufferung)

Modell: Eingabevollständige Produkttransitionssysteme, Spurmengen, Stromfunktionen

Kommunizierende Prozesse: Sequentielle Prozesse

Ziel: Modell für das Verhalten von stark gekoppelten Systemen

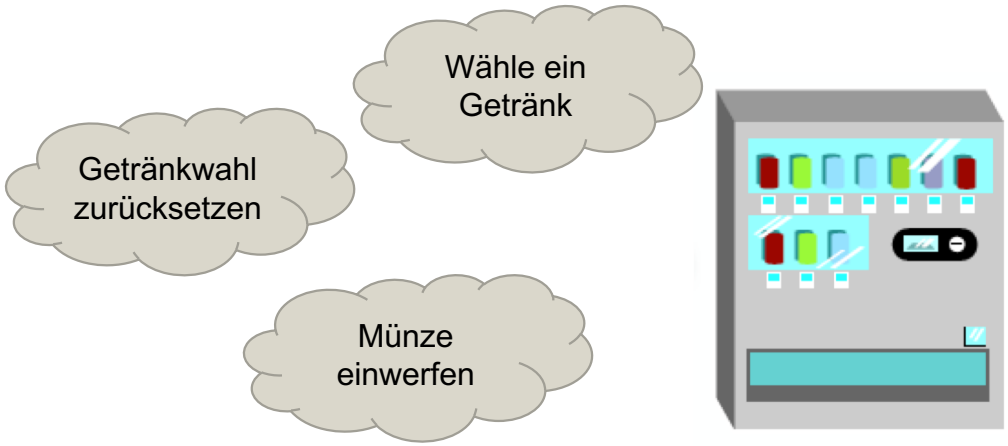
- Prozesse als interagierende Einheiten
- Definition von sequentiellem Verhalten
- Element für die Kontrollflusssteuerung

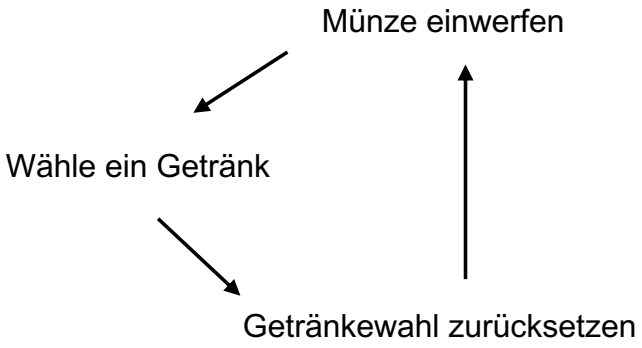
Konzept: Sequentieller interagierender Prozess

Notation: (T)CSP (Communicating Sequential Processes)

Modell: Markierte Transitionssysteme

Konzept: Prozess





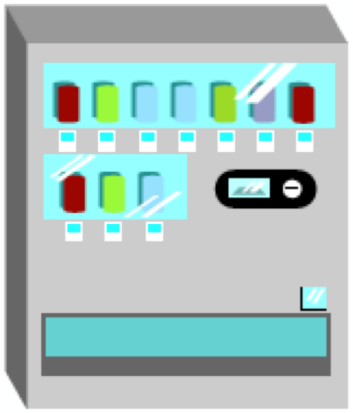
Ziel: Definition von „Verhaltensblöcken“

Prozess: (Abstrakte) Ausführungseinheit

- Interaktionen: Menge atomarer, gemeinsamer Aktionen eines Prozesses und seiner Umgebung
- Ausführung: Sequenzen von Interaktionen eines Prozesses
- Verhalten: Menge von Ausführungen

Sequentieller Prozess: Ausführung eines interagierenden sequentiellen Programms

Notation: Prozessterm



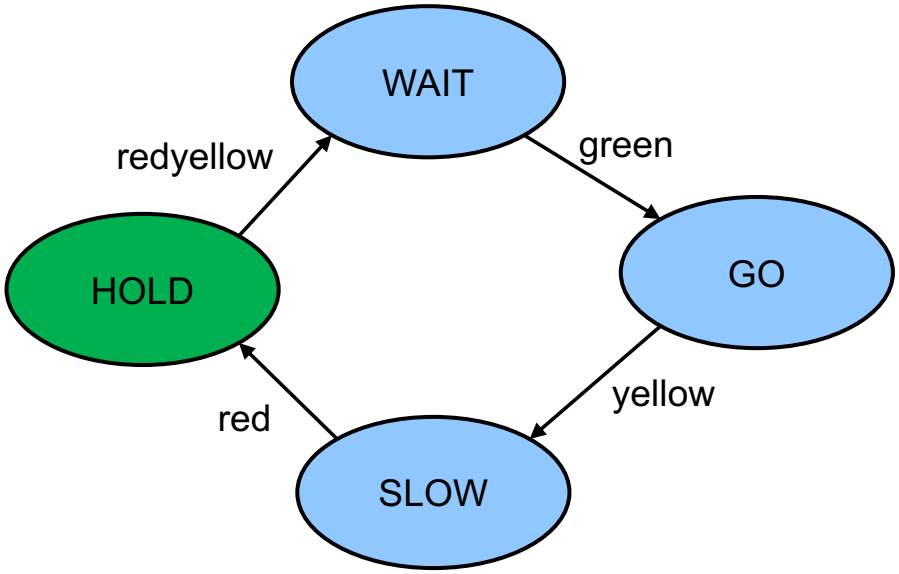
Machine = coin → Wait
Wait = (coffee-button → Coffee) |
(tea-button → Tea)
Coffee = coffee → Machine
Tea = tea → Machine

Ziel: Prozessbeschreibung mittels textueller Notation als Basis algebraischer Regeln zur Verhaltensmodellierung

Ansatz: Beschreibung von Prozessstrukturen und Identifikation gleichen Verhaltens

- Basisprozesse: Primitive Prozesse als Grundbausteine
- Operatoren: Konstruktoren zum Aufbau komplexer aus einfacheren Prozessen
- (Gleichungs-)Regeln: Regeln zur Identifikation äquivalenter Prozesse

Konzept: Alphabet



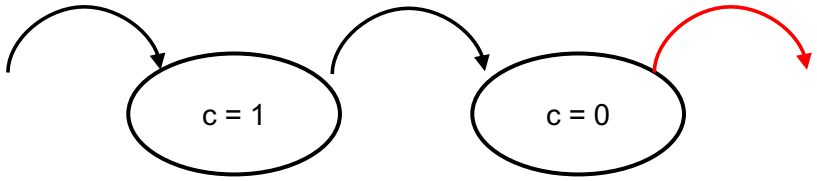
Ziel: Beschreibung der Interaktionen zwischen Prozess und Umgebung

Alphabet: Menge der für den Prozess generell möglichen Interaktionen

- Implizit: Aktionen der Konstituenten des Prozessterms (z.B. Präfix)
- Explizit: Aktionen in expliziten Annotationen definiert (z.B. $STOP_{\{green,red\}}$)

Beispiel: Alphabet des Prozesses HOLD ist { redyellow, green, yellow, red }

Notation: Termination



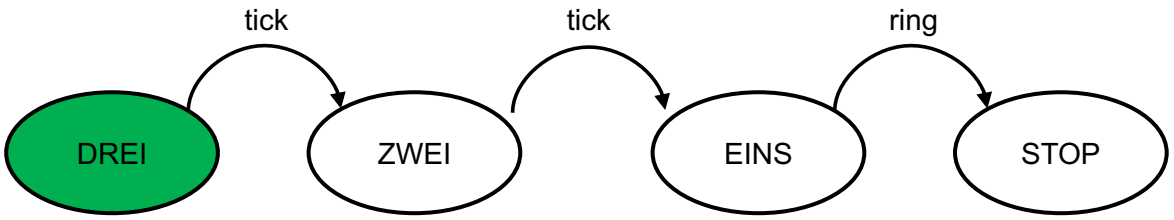
Ziel: Beschreibung eines terminierenden Prozesses

Termination: Prozess, der alle Interaktionen blockiert

- $STOP_A$: Prozess der sofort terminiert und damit alle Interaktionen blockiert
- Alphabet: A als mögliche Interaktionen des Prozesses

Notation: Process := $STOP_A$

Notation: Aktionsausführung



Ziel: Beschreibung der Initialaktion eines sequentiellen Prozesses

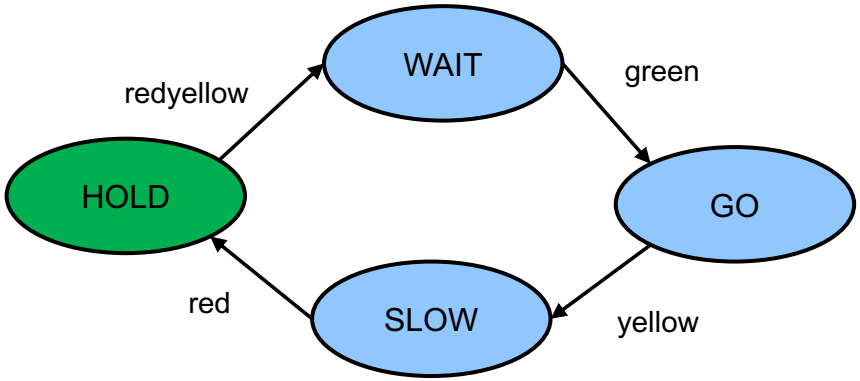
Aktionsvorschaltung: Prozessterm zur Vorschaltung einer Aktion *a* vor Prozess *P*

- Aktion *a*: Initial auszuführende Aktion („Präfix“)
- Prozess *P*: Nach Aktion a auszuführender Prozess
- Alphabet: Alphabet(P) ∪ {a}

Notation: Process := action → Process

Beispiel: DREI = tick → ZWEI, ZWEI = tick → EINS, EINS = ring → STOP

Konzept: Rekursion



Ziel: Beschreibung Verhaltens eines sequentiellen(nichtterminierenden) Prozesses

Rekursion: Definition eines Prozessterms für *P* unter Verwendung des Prozesses

- Einfache Rekursion: Eine Definition mit definierendem / anwendendem Auftreten
- Verschachtelte Rekursion: Mehrere, wechselseitige abhängige Prozessdefinitionen
- Fixpunkt: Explizite Definition einer Prozessvariablen X
- Alphabet: Explizite Definition

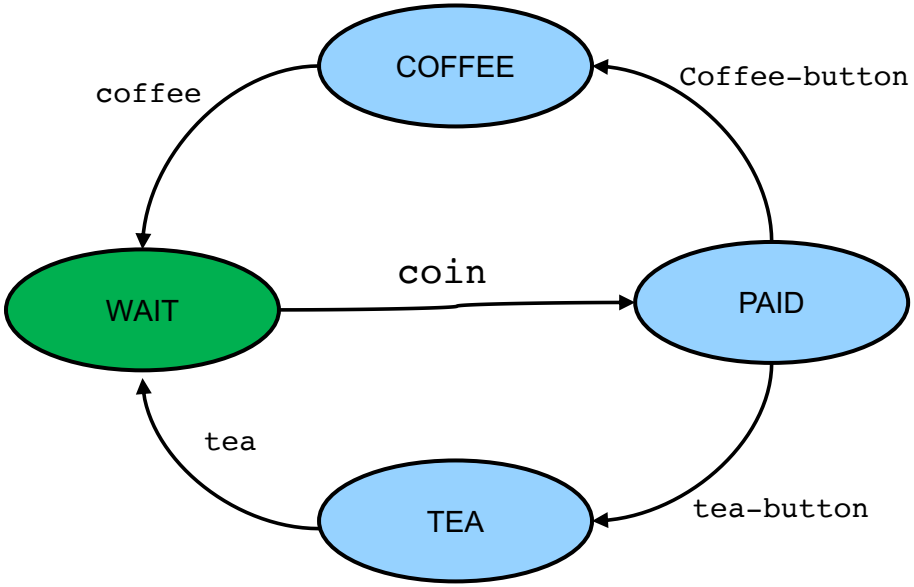
Notation: Process := Term(Process) bzw. Prozess = μ X:Alphabet . Term(X)

Beispiel:

- HOLD = redyellow → green → yellow → red → HOLD
- HOLD = redyellow → WAIT, WAIT = green → GO, GO = yellow → SLOW,

SLOW = red → HOLD

Notation: Auswahl



Ziel: Beschreibung von alternativem Verhalten eines sequentiellen Prozesses

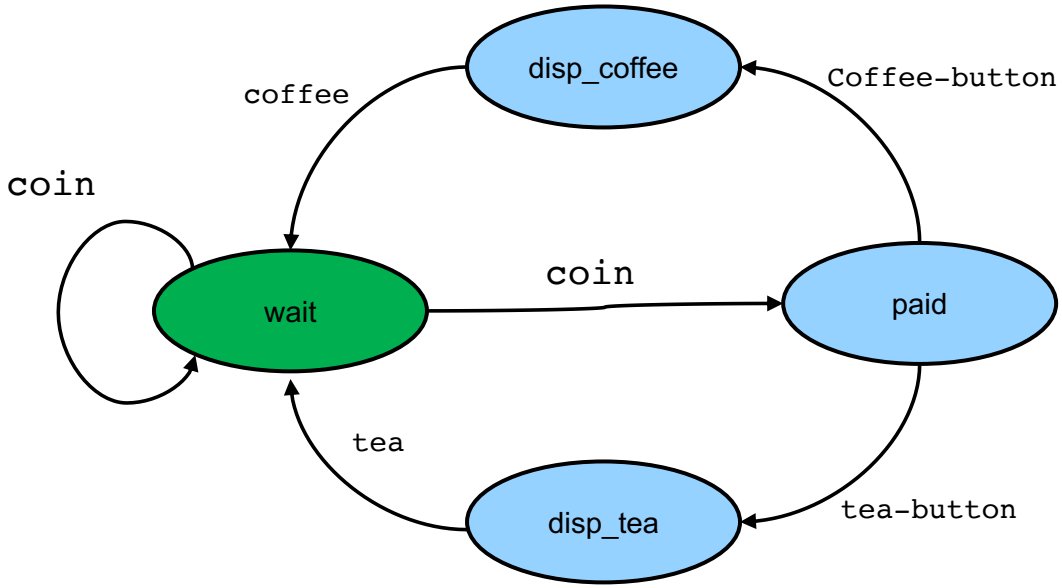
Auswahl: Prozessterm zur Kombination zweier alternativer Prozesse *P* und *Q*

- Prozess kann die Aktionen ausführen, die *P* oder *Q* ausführen können
- Prozess lehnt nur Aktionen ab, die sowohl *P* und *Q* ablehnen
- Alphabet: Alphabet(*P*) ∪ Alphabet(*Q*)

Notation: *Process := Process* | *Process*

Beispiel: PAID = (tea-button → TEA) | (coffee-button → COFFEE)

Notation: Nichtdeterminismus



Ziel: Beschreibung von alternativem Verhalten eines sequentiellen Prozesses

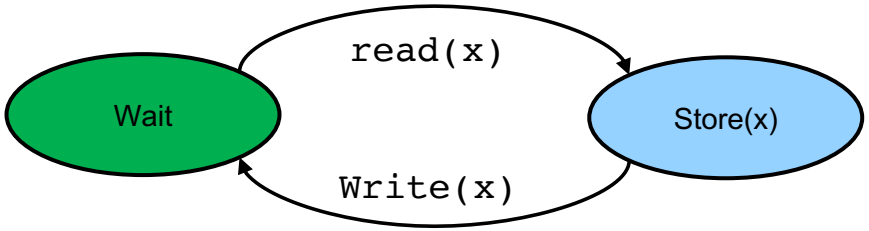
Nichtdeterminismus: Prozessterm kombiniert zwei alternative Prozesse *P* und *Q*

- Prozess kann sich wie *P* oder *Q* verhalten
- Prozess kann Aktionen von *P* oder *Q* zurückweisen
- Alphabet: Alphabet(*P*) ∪ Alphabet(*Q*)

Notation: *Process := Process* ∏ *Process*

Beispiel: WAIT = (coin → WAIT) ∏ (coin → PAID)

Notation: Parameter



Ziel: Beschreibung von parametriertem Verhalten eines Prozesses

Parameter: Prozess abhängig von formalen Parameter x mit Domäne D

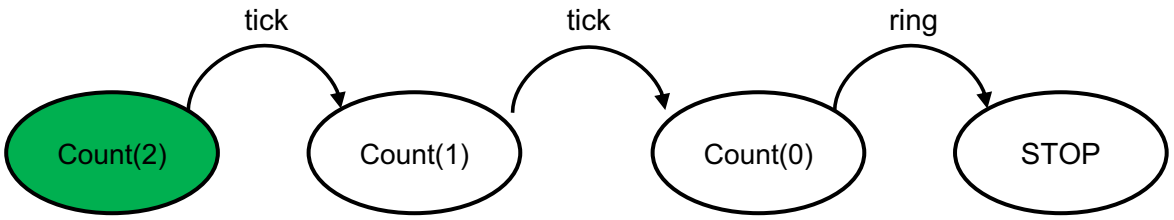
- Parametrierter Prozess: Definition mit allen Belegungen instantiiert
- Parametrierte Aktion: Parameter mit Belegung der Definition instantiiert
- Alphabet: Alphabet aller Instantiierungen

Notation: *action := action(x); Process := Process(x)*

Beispiel: Wait = read(x) → Store(x), Store(x) = write(x) → Wait entspricht

Wait = read(0) → Store(0), Store(0) = write(0) → Wait, Wait = read(1) →

Konzept: Wächter



Ziel: Definition des parametrierten Kontrollflusses der (Inter-)Aktionen

Bewachte Aktion: Prozessterm, abhängig von Bedingungen c und Prozess P

- Gültige Bedingung: Ist *c* gültig, entspricht der Prozess P
- Ungültige Bedingung: Ist *c* ungültig, entspricht der Prozess STOP
- Alphabet: Alphabet(P)

Notation: Process := Condition ▷ Process

Beispiel: Count(x) = ((x > 0) ▷ (tick → Count(x-1))) | ((x = 0) ▷ (ring → Halt))

Kommunizierende Prozesse: Nebenläufige Prozesse

Ziel: Modell für das Verhalten von stark gekoppelten Systemen

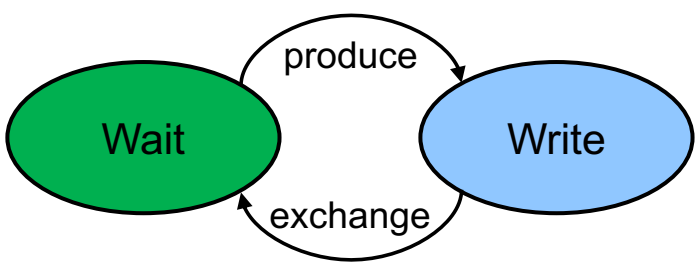
- Komposition von synchronisierten Prozessen
- Verbergen von internen Aktionen

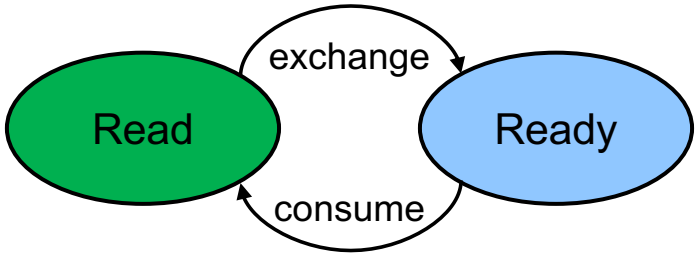
Konzept: Nebenläufige Ausführung, synchronisierende Kommunikation

Notation: (T)CSP (Communicating Sequential Processes)

Modell: Prozessalgebren

Notation: Komposition





Ziel: Modellierung komplexer nebenläufiger Systeme

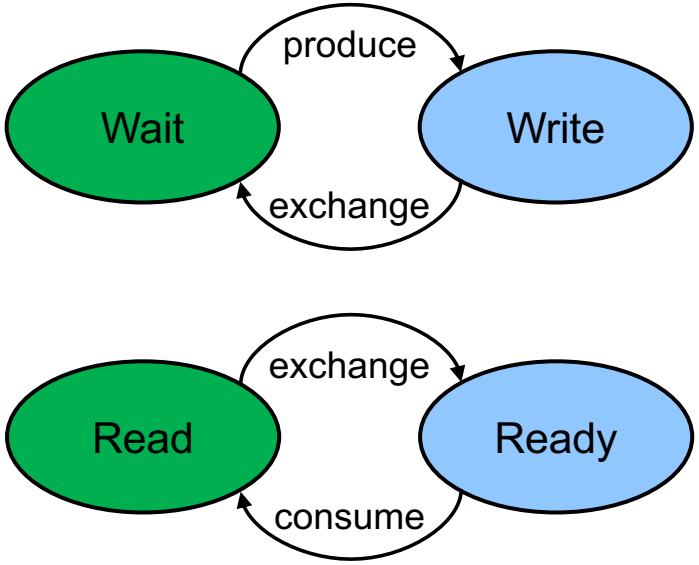
Komposition: Prozessterm komponiert Prozesse *P* und *Q*

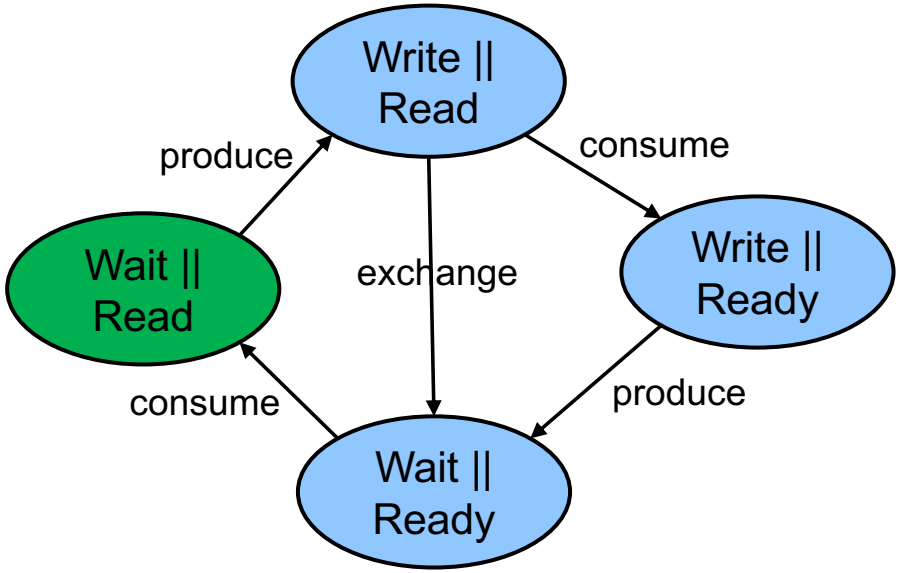
- Prozess: Nebenläufige Ausführung der Prozesse *P* und *Q*
- Alphabet: Alphabet(P) ∪ Alphabet(Q)

Notation: Process := Process || Process

Beispiel: ProCon = Wait || Read, Wait = produce → exchange → Wait, Read= exchange → consume → Read

Konzept: Synchronisierte Prozesse





Ziel: Modell des Verhaltens nebenläufiger synchron kommunizierender Prozesse

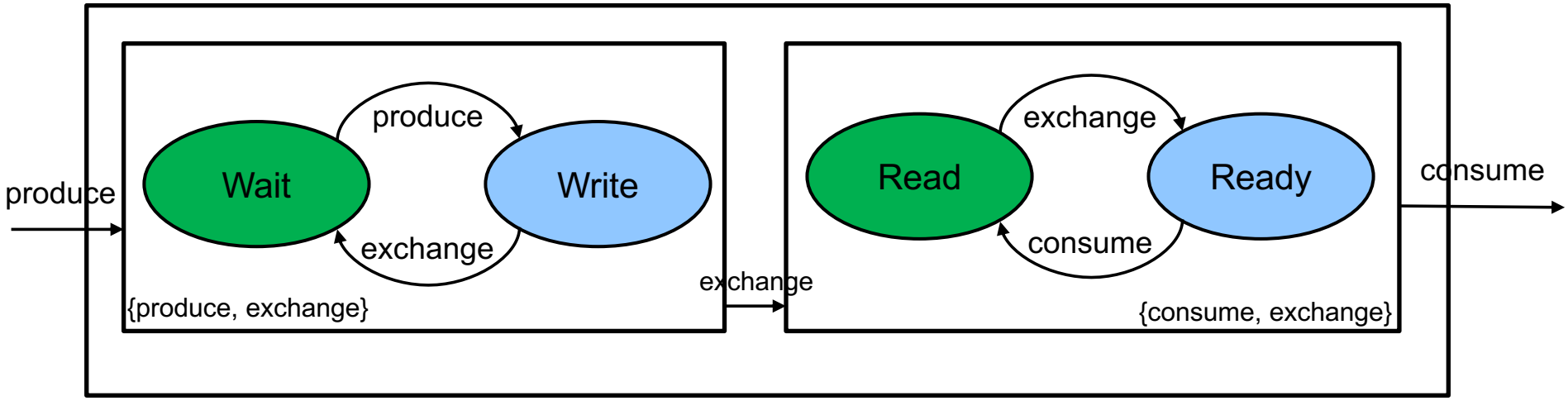
Synchronation: Nebenläufige Ausführung mit simultaner Interaktion

- Gemeinsame Aktionen: Aktionen des gemeinsamen Alphabets simultan ausgeführt
- Getrennte Aktionen: Aktionen der einzelnen Alphabete sequentiell ausgeführt

Modell: Prozessalgebra/Synchronisiertes markiertes Produkttransitionsystem

Beispiel: Pro = produce → exchange →Pro, Con = exchange → consume → Con, Pro||Con = WR, WR = produce → TR, TR = exchange → WY, WY = consume → WR| exchange → TY, TY= consume → TR

Notation: Verbergen



Ziel: Unterscheidung interne (zwischen Prozessen) und externe (zwischen Prozess und Umgebung) Interaktionen

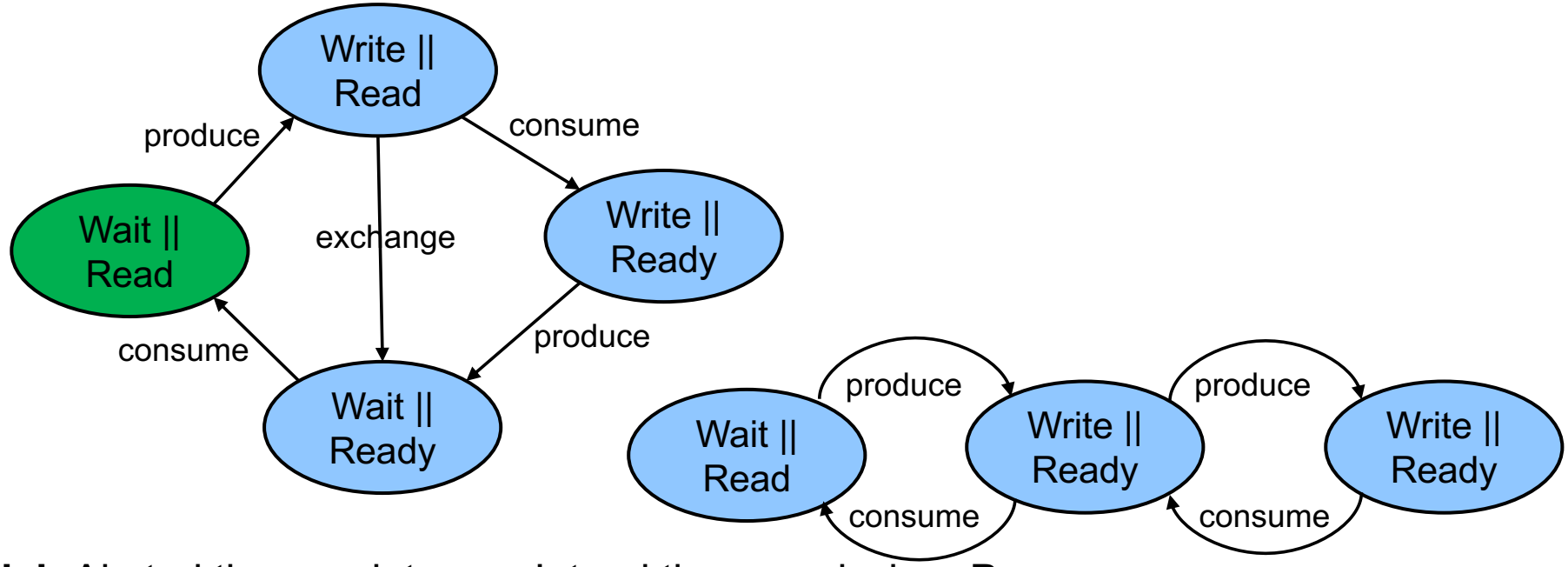
Verbergen: Abstraktion von internen Interaktionen *H* eines Prozesses *P*

- Abstraktion: Interaktionen von *H* werden für Umgebung unsichtbar
- Alphabet: Alphabet(*P*) \ *H*

Notation: *Process* := *Process* \ *Alphabet*

Beispiel: Alphabet von ProCon \ {exchange} ist { produce, consume }

Konzept: Verbergen



Ziel: Abstraktion von internen Interaktionen zwischen Prozessen

Verbergen: Verhalten eines Prozesses abzüglich interner Aktionen

- Interne Aktionen: Finden unmittelbar statt, sobald möglich
- Externe Aktionen: Finden synchronisiert mit Umgebung statt

Beispiel: AbsProCon = ProCon \ {exchange}

AbsProCon = produce → Buffer, Buffer = (consume → AbsProCon) |

(produce → consume → Buffer)

Model: Prozessalgebra

Ziel: Definition von Prozessen und Herleitung von deren Verhaltensäquivalenz

Prozessalgebra: Kalkül aus Prozesstermen und Gesetzen

- Termsprache: Basisterme und Konstruktoren von Prozesstermen
- Algebraische Gesetze: Gleichheitsregeln zur Identifikation äquivalenter Prozesse
- Äquivalenzherleitung: Transformation äquivalenter Terme in einander mittels Regeln

Beispiel:

– $P \mid STOPA = P$

– $(a \rightarrow P) \,||\, (a \rightarrow Q) = a \rightarrow (P \,||\, Q)$

Model: Prozessalgebra

Weitere Regelbeispiele: (αP =αQ = A)

- P | STOPA = P
- false ▷ P = STOP
- μX : A . Term(X) = Term(μX : A . Term(X))
- P = μX:A . Term(X) für P = Term(P)
- P || Q = Q || P
- P || (Q || R) = (P || Q) || R
- P || STOPA = STOPA
- (a → P) || (a → Q) = a → (P || Q)
- (a → P) || (b → Q) = (a → (P || (b → Q)) | (b → ((a → P)||Q)) (falls a ≠ b)
- STOPA \ H = STOPA \ H
- (a → P) \ { a } = P \ { a }

Beispiel: Prozessalgebra

Beispiel: Herleitung von S = T || U

für S = μ P:A. (a → P | b → P), T = (μ Q:{a}. a → Q), U = (μ R:{b}. b → R)

mit A = {a,b}

Herleitung: Sei V = T || U

Dann: V

= T || U

= (μ Q:{a}. a → Q) || (μ R:{b} . b → R)

= (a → (μ Q:{a}. a → Q)) || (b → (μ R:{b}. b → R))

= ((a → T) || (b → U))

= (a → (T || (b → U))) | (b → ((a →T) || U))

= (a → (T ||U)) | (b → (T||U))

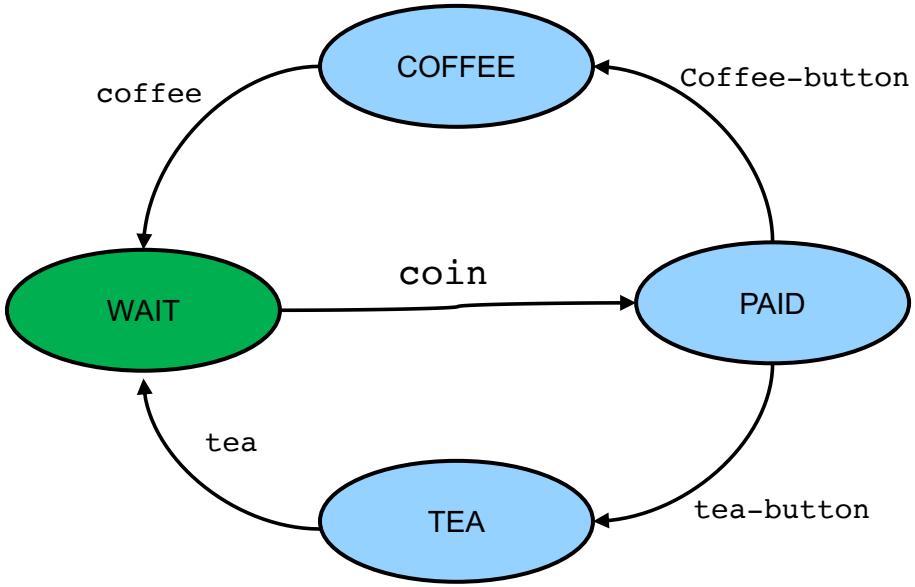
= (a → V) | (b → V)

Und damit: V = μ P:A. (a → P | b → P)

Sowie: S = V

Sowie: S = T || U

Semantik: Prozessterme



Machine = coin → Wait
Wait = (coffee-button → Coffee) |
(tea-button → Tea)
Coffee = coffee → Machine
Tea = tea → Machine

Ziel: Interpretation von Prozesstermen

Interpretation: Verschiedene Möglichkeiten

1. Überführung in markiertes Transitionssystem (Zustände als Termidentifikatoren, markierte Transition als Aktionsvorschaltung)
2. Überführung Menge von beobachtbare Abläufe (direkt oder über 1)
3. Direkte Anwendung Prozessalgebra

Alle drei Interpretationen erlauben den Vergleich syntaktisch unterschiedlicher Systeme auf gleiches Verhalten

Modelle reaktiver Systeme

Modellarten:

- **Operationales Modell:** Spezifikation als abstrakte Maschine
  - Konzepte: Zustand, (markierte) Transition, Transitionssystem
  - Verifikation: (Bi-)Simulation
  - Beispiel: (Gezeitete) Automaten
- **Denotationales Modell:** Spezifikation über beobachtbares Verhalten
  - Konzepte: Interface, Ereignis, (beobachtbarer) Ablauf
  - Verifikation: Verhaltensinklusion
  - Beispiel: Spur- bzw. ablaufbasierte Modelle
- **Algebraisches Modell:** Spezifikation als syntaktische Formeln
  - Konzepte: Prozess, Aktion, Auswahl, Rekursion
  - Verifikation: Termäquivalenz
  - Beispiel: Prozessterme