Prozess Algebra (PA)

- Modellierung *nebenläufiger* Systeme
  - Calculus of Communicating Systems (CCS) [Milner80]
  - Communicating Sequential Processes (CSP) [Hoare85]
  - genauer: **asynchron** kommunizierende Prozesse (Protokolle/Software)
- Synthese: Prozess Algebra (PA) als Programmiersprache (z.B. Occam, Lotos)
- Verifikation von (abstrakteren) PA Modellen ist einfacher
- **Theorie:** Mathematische Eigenschaften nebenläufiger Systeme
  - Wie kann man nebenläufige Systeme vergleichen?
  - Simulation, Bisimulation, Beobachtbarkeit, Divergenz (⇒ Systemtheorie 1)

PA Gleichungen (PAE)

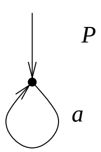
- Rechtslineare Grammatik = Reguläre Sprache = Chomsky 3 Sprache

  Grammatik G: $N = \varepsilon \mid aM \mid bM$ $\quad M = cN \mid dN$ $\quad$ Startsymbol N

  ⇒ Sprache $L(G) = ((a \mid b)(c \mid d))^*$ (als regulärer Ausdruck)
- Syntax bei PA:
  - gleiche Idee: Gleichungen über Nichterminalen = Prozesse
  - Konkatenation nicht durch Hintereinanderschreiben sondern mit '.' Operator
  - Auswahl dargestellt durch '$+$' Operator (nicht durch '$|$')
- Semantik
  - nur interessiert an den möglichen Sequenzen (= Ereignisströme)

Konkatenation

Graphische Darstellung



$P = a.P$

Gleichung

$R_.$ $\quad \dfrac{}{a.P \xrightarrow{a} P}$

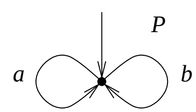
Regel Operationale Semantik
(hier ist P eine Metavariable)

'.' Operator bedeutet sequentielle Konkatentation

Auswahl

Graphische Darstellung

R^1_+ $\quad \dfrac{P \xrightarrow{a} P'}{(P+Q) \xrightarrow{a} P'}$



$P = a.P + b.P$

R^2_+ $\quad \dfrac{Q \xrightarrow{a} Q'}{(P+Q) \xrightarrow{a} Q'}$

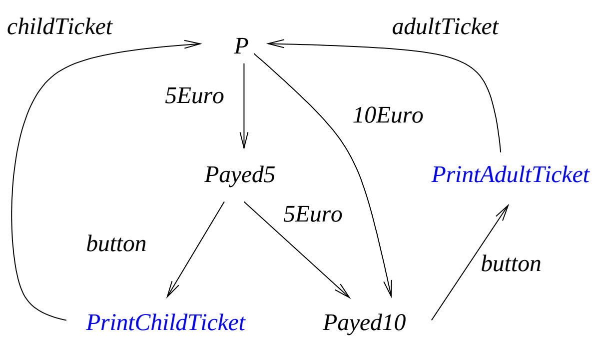
Gleichung

Regeln Operationale Semantik
(hier sind P, Q Metavariablen)

'$+$' Operator bedeutet nicht-deterministische Auswahl

Beispiel Eintrittskarten-Automat

$$
\begin{aligned}
P &= \mathit{5Euro}.\mathit{Payed5} + \mathit{10Euro}.\mathit{Payed10} \\
\mathit{Payed5} &= \mathit{button}.\mathit{childTicket}.P + \mathit{5Euro}.\mathit{Payed10} \\
\mathit{Payed10} &= \mathit{button}.\mathit{adultTicket}.P
\end{aligned}
$$



Labelled Transition Systems (LTS)

- LTS als **operationalen Semantik** von PAE
- fast wie Automaten, aber …
  - keine Finalzustände bzw. im Prinzip sind alle Zustände Finalzustände
  - man ist nur an möglichen Ereignissequenzen interessiert
- LTS $A = (S, I, \Sigma, T)$ mit
  - Zustandsmenge S
  - Aktionen Σ
  - Übergangsrelation $T \subseteq S \times \Sigma \times S$ definiert durch operationale Regeln
  - Anfangszuständen $I \subseteq S$

Syntaktische Einschränkungen

- Divergente Selbstzyklen
  - $P = a.P + P$ ist **keine** gültige PAE
  - es gibt keine ε-Übergänge im Gegensatz zu EA
    (ε ist keine Aktion, da es ja “keine Zeit” braucht)
- Vermeidung von Selbstzyklen
  - Term T heißt **bewacht** bzw. **guarded** wenn T nur in der Form $a.T$ vorkommt
    (wobei a natürlich unterschiedlich für jedes Vorkommen von T sein kann)
  - einfachste Einschränkung:

    Prozessvariablen auf rechter Seite (RHS) einer PAE sind bewacht
  - oder komplexer: jeder “Zyklus” beinhaltet mindestens eine Aktion

Daten in Prozess Algebra

- Aktionen und Zustände können **parametrisiert** sein
  - somit auch parametrisierte Gleichungen
- voriges Beispiel im neuen Gewand ($x \in \{5, 10\}$):

$$
\begin{aligned}
P &= \mathit{euro}(x).\mathit{Payed}(x) \\
\mathit{Payed}(5) &= \mathit{button}.\mathit{print}(\mathit{childTicket}).P + \mathit{euro}(5).\mathit{Payed}(10) \\
\mathit{Payed}(10) &= \mathit{button}.\mathit{print}(\mathit{adultTicket}).P
\end{aligned}
$$

- möglicherweise zusätzliche Operationen auf den Daten erlaubt:

$$
\mathit{Payed}(X) = \mathit{euro}(Y).\mathit{Payed}(X+Y) + \mathit{button}.\mathit{ticket}(X).P
$$

  - damit insgesamt Beschreibung von *unendlichen Systemen* möglich
  - wird dadurch auch zur echten Programmiersprache

Bedingungen

$$R_{\text{then}} \quad \frac{P \xrightarrow{a} P'}{\textbf{if } B \textbf{ then } P \textbf{ else } Q \xrightarrow{a} P'} \quad B$$

$$R_{\text{else}} \quad \frac{Q \xrightarrow{a} Q'}{\textbf{if } B \textbf{ then } P \textbf{ else } Q \xrightarrow{a} Q'} \quad \neg B$$

(und ähnliche Regeln für **if-then** alleine)

$$\begin{aligned} Payed(X) &= euro(Y).Payed(X+Y) + button.Print(X) \\ Print(X) &= \textbf{if } (X=5) \textbf{ then } childTicket.P + \textbf{if } (X=10) \textbf{ then } adultTicket.P \end{aligned}$$

Parallel-Operator

Synchronisation durch Rendezvous wie in CSP

$$\Theta \subseteq \Sigma$$

$$R_{||_\Theta} \quad \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{a} Q'}{P \,||_\Theta\, Q \xrightarrow{a} P' \,||_\Theta\, Q'} \quad a \in \Theta \qquad \text{Rendezvous}$$

$$R^1_{||_\Theta} \quad \frac{P \xrightarrow{a} P'}{P \,||_\Theta\, Q \xrightarrow{a} P' \,||_\Theta\, Q} \quad a \notin \Theta \qquad \text{Interleaving}$$

$$R^2_{||_\Theta} \quad \frac{Q \xrightarrow{a} Q'}{P \,||_\Theta\, Q \xrightarrow{a} P \,||_\Theta\, Q'} \quad a \notin \Theta \qquad \text{Interleaving}$$

Beim Rendezvous ist Sender und Empfänger nicht genauer spezifiziert!

$$R_{||} \quad \frac{P \,||_\Theta\, Q \xrightarrow{a} P' \,||_\Theta\, Q'}{P \,||\, Q \xrightarrow{a} P' \,||\, Q'} \quad \Theta = \Sigma(P) \cap \Sigma(Q)$$

$\Sigma(P)$ ist die Teilmenge der Aktionen von Σ die in P syntaktisch vorkommen

Eigenschaften des Parallel-Operators

Fakt $||$ ist kommutativ: $P \,||\, Q \xrightarrow{a} P' \,||\, Q'$ gdw. $Q \,||\, P \xrightarrow{a} Q' \,||\, P'$

Beweis folgt unmittelbar aus den Regeln

Fakt $||$ ist assoziativ

Beweis: Sei $P = P_1 \,||\, (P_2 \,||\, P_3)$, $P' = P'_1 \,||\, (P'_2 \,||\, P'_3)$, $Q = (P_1 \,||\, P_2) \,||\, P_3$, $Q' = (P'_1 \,||\, P'_2) \,||\, P'_3$

Zu Zeigen: $P \xrightarrow{a} P' \quad \Leftrightarrow \quad Q \xrightarrow{a} Q'$

Genauer Beweis: 8 Fälle der Zugehörigkeit von $a \in \Sigma(P_i)$ für beide Richtungen.

Intuition:

1. $a \in \Sigma(P_i) \Rightarrow P_i \xrightarrow{a} P'_i$
2. P_i mit $a \notin \Sigma(P_i)$ ändern sich nicht ($P'_i = P_i$)
3. dasselbe gilt für jede "parallele Zusammenschaltung" von P_i

Folgerungen der Eigenschaften des Parallel-Operators

- Klammerung bei $||$ kann weggelassen werden:

  $P \,||\, (Q \,||\, R)$ verhält sich wie $(P \,||\, Q) \,||\, R$ verhält sich wie $P \,||\, Q \,||\, R$

- weiter kann Anordnung ignoriert werden

  $P \,||\, Q \,||\, R$ verhält sich wie $P \,||\, R \,||\, Q$ verhält sich wie $Q \,||\, P \,||\, R$ etc.

- Parallel-Schaltung $\underset{i \in J}{||}\, P_i$ bel. Prozesse P_i über Indexmenge J:

$$R_{||} \quad \frac{\forall P_i, a \in \Sigma(P_i) \quad P_i \xrightarrow{a} P'_i \qquad\qquad \forall P_i, a \notin \Sigma(P_i) \quad P'_i = P_i}{||P_i \quad \xrightarrow{a} \quad ||P'_i} \qquad \exists P_i \quad P_i \xrightarrow{a} P'_i$$

Hiding

- Verstecken bzw. Abstraktion von internen, **unbeobachtbaren** Aktionen

- Abstraktion zur "stillen" Aktion τ
  - Annahme: $\tau \notin \Sigma$
    - formal betracht hat man nun Aktionen $\Sigma \dot{\cup} \{\tau\}$
    - damit kann auch nie auf τ synchronisiert werden
  - τ verbraucht trotzdem einen Zeitschritt

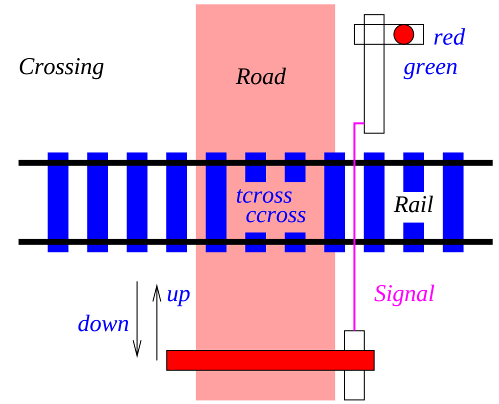
$$R^{\notin}_{\backslash} \quad \frac{P \xrightarrow{a} Q}{P\backslash\Theta \xrightarrow{a} Q\backslash\Theta} \quad a \notin \Theta \qquad\qquad R^{\in}_{\backslash} \quad \frac{P \xrightarrow{a} Q}{P\backslash\Theta \xrightarrow{\tau} Q\backslash\Theta} \quad a \in \Theta$$

- typische Verwendung für interne Synchronisationen $R = (||_{i=1}^{n} Q_i)\backslash\{x_1,\ldots,x_n\}$

Beispiel Bahnübergang

[BradfieldStirling]

$$\begin{aligned}
Road &= car.up.ccross.down.Road \\
Rail &= train.green.tcross.red.Rail \\
Signal &= green.red.Signal + up.down.Signal \\
Crossing &= (Road \,||\, Rail \,||\, Signal)\backslash\{green, red, up, down\}
\end{aligned}$$



Linking

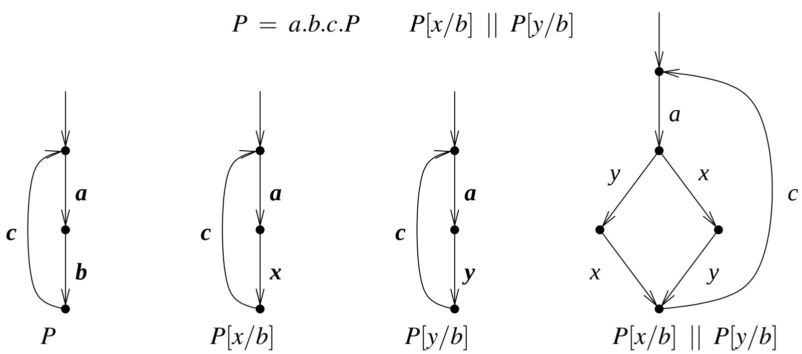
Linking als Substitution von Aktionen

$$R_{[\,]} \quad \frac{P \xrightarrow{a} Q}{P[b/a] \xrightarrow{b} Q[b/a]}$$

Beispiel: $(a.P)[b/a] \xrightarrow{b} P[b/a]$

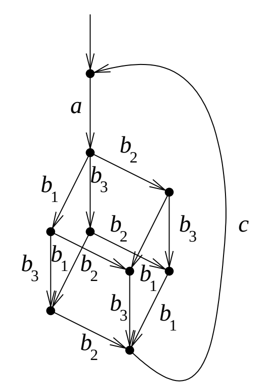
wird benötigt um Prozesse zusammenzubinden oder Templates zu instanziieren:

$P = a.b.c.P \qquad P[x/b] \;||\; P[y/b]$



Parametrisierung von Linking

$P = a.b.c.P \qquad\qquad \Big\|_{i=1}^{3} P[b_i/b]$



Fragestellung Milner's Scheduler

- klassisches Beispiel aus der Prozessalgebra
  - Modellierung eines Round-Robin-Schedulers **möglichst allgemein**
- Scheduling von n Prozessen $\quad || P_i \quad$ mit $\quad P = a.z.b.P \quad$ und $\quad P_i = P[a_i/a, z_i/z, b_i/b]$
  - a modelliert Starten eines Durchlaufes eines Prozesses
  - z modelliert interne Aktion bzw. interne Aktionen
  - b modelliert Ende des Durchlaufes eines Prozesses
- **Einschränkungen**:
  - Prozesse werden im Round-Robin-Stil gestartet in der Reihenfolge $P_1, P_2, \ldots$
  - Prozess kann nicht zweitesmal gestartet werden ohne beendet worden zu sein
  - es wird nichts über die Reihenfolge der b_i gesagt!

Falsche Lösung Milner's Scheduling

- Lösungsansatz: Proxy für jeden zu kontrollierenden Prozess
- Zerlegung des Schedulers R' in Token-Ring von n parallelen zyklischen Prozessen Q'
- jedes Q'_i kontrolliert Starten (a_i) und Beenden (b_i) von P_i, …
- … übergibt Ausführungserlaubnis x_i an nächsten Q'_{i+1} …
- und wartet dann auf Ausführungserlaubnis x_{i-1} vom vorigen Q'_{i-1} im Ring

$$
\begin{aligned}
Q' &= a.x.b.y.Q' \\
Q'_1 &= Q'[a_1/a,\ x_1/x,\ b_1/b,\ x_n/y] \\
Q'_i &= (y.Q')[a_i/a,\ x_i/x,\ b_i/b,\ x_{i-1}/y] \qquad i \in \{2,\ldots,n\} \\
R' &= \mathop{||}_{i=1}^{n} Q'_i
\end{aligned}
$$

Richtige Lösung Milner's Scheduling

- falsche Lösung akzeptiert folgende legale Sequenz **nicht**:
  - Beenden von P_2 vor P_1:

$$a_1 a_2 b_2 b_1 \ldots$$

- Entkopplung des Beendens (b) und der Berechtigungsannahme (y)

$$
\begin{aligned}
Q &= a.x.\boxed{(b.y + y.b)}.Q \\
Q_1 &= Q[a_1/a,\ x_1/x,\ b_1/b,\ x_n/y] \\
Q_i &= (y.Q)[a_i/a,\ x_i/x,\ b_i/b,\ x_{i-1}/y] \qquad i \in \{2,\ldots,n\} \\
R &= \mathop{||}_{i=1}^{n} Q_i
\end{aligned}
$$

- Implementierung durch Warten auf zwei unterschiedliche Nachrichten

Unterschiede in CCS

- Aktionen: $\Sigma \,\dot\cup\, \overline{\Sigma} \,\dot\cup\, \{\tau\}$
  - gestrichene Aktionen Ausgaben, ungestrichene Eingaben
- anderes Hiding-Prinzip (Doppel-Schrägstrich zur syntaktischen Unterscheidung)

$$R_{\backslash\backslash} \qquad \frac{P \xrightarrow{a} Q}{P \backslash\backslash \Theta \xrightarrow{a} Q \backslash\backslash \Theta} \qquad a \notin \Theta \cup \overline{\Theta}$$

- paarweise **explizite** Synchronisation

$$R_{|||} \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \,|||\, Q \xrightarrow{\tau} P' \,|||\, Q'} \qquad a \in \Sigma \cup \overline{\Sigma}$$

$$R^1_{|||} \qquad \frac{P \xrightarrow{a} P'}{P \,|||\, Q \xrightarrow{a} P' \,|||\, Q} \qquad\qquad R^2_{|||} \qquad \frac{Q \xrightarrow{a} Q'}{P \,|||\, Q \xrightarrow{a} P \,|||\, Q'}$$

Vergleich CSP und CCS am Beispiel Bahnübergang

$$\begin{aligned}
Road &= car.up.ccross.down.Road \\
Rail &= train.green.tcross.red.Rail \\
Signal &= green.red.Signal + up.down.Signal \\
Crossing &= (Road \,||\, Rail \,||\, Signal) \backslash \{green, red, up, down\}
\end{aligned}$$

bzw. in CCS

$$\begin{aligned}
Road &= car.up.\overline{ccross}.\overline{down}.Road \\
Rail &= train.green.\overline{tcross}.\overline{red}.Rail \\
Signal &= \overline{green}.red.Signal + \overline{up}.down.Signal \\
Crossing &= (Road \,|||\, Rail \,|||\, Signal) \backslash\backslash \{green, red, up, down\}
\end{aligned}$$

Weitere Varianten

- Orginalversion Kanäle mit Daten bei CSP
  - Eingabe: $channel\,?\,datain$, Ausgabe: $channel\,!\,dataout$
- π-Kalkül nach [MilnerParrowWalker]
  - Kanäle/Verbindungen werden selbst Daten
  - Beispiel: $TimeAnnounce = ring(caller).\overline{caller}(CurrentTime).\overline{hangup}.TimeAnnounce$
- Probabilistisches Verhalten
  - Übergänge sind mit einer Übergangswahrscheinlichkeit versehen
- Prozess Algebra mit Zeit
  - Übergänge *brauchen* explizit angegebene Zeit