Fahrplan Pi Kalkül Vorstellung:

1. Was ist der Pi Kalkül
    - Beschreibung nebenläufiger Prozesse, die sich während der Laufzeit verändern
    - es geht um die Kommunikation unabhängiger Prozesse
        - über Kanäle werden ~~Daten~~ Namen ausgetauscht (Message-Passing-Modell, nur Nachrichten werden ausgetauscht)

2. Syntax

N = abzählbar unendliche Menge von Namen.
c, x element N

Zusammenfassung der Syntax:

P ::=		π.P	(Aktion)
	|	P1 | P2	(Parallele Komposition)
	|	!P	(Replikation)
	|	0	(inaktiver Prozess)
	|	vx.P	(Restriktion)
π ::=		x(y)	(Input)
	|	$\bar{x}$<y>	(Output)

Prozesse P, Q, Prozesse können neben Terme des Pi-Kalküls auch echte Programme darstellen.

Nullprozess 0

Eingabepräfix c(x).P
Erklärung: Auf dem Kanal c wird eine Eingabe erwatet. Nach dem Eingang einer Nachricht wird x substituiert durch den empfangenen Namen im Prozess P, sofern x frei ist. Solange keine Nachricht empfangen wird, blockiert c und P kann nicht ausgeführt werden.

Beispiel: Methode „p1won“ als Kanal, erwartet boolean Wert, Prozess: Print „P1 won“ if true, Print „P2 won“ if false, bool b wird duch den empfangenen ersetzt, z.B. durch true

Darstellung in Psedudo-Code	Darstellung in Pi-Kalkül

`void p1won(bool b){`
`    if (b) print(“P1 won”)`
`    else print(“P2 won”)`
`}`
	p1won(boolean).Print

Ausgabepräfix $\bar{c}$<x>.P

Erklärung: Auf dem Kanal c wird eine Nachricht ausgegeben. Solange es keinen Empfänger für die Nachricht gibt, blockiert c und P kann nicht ausgeführt werden.

Beispiel: Methode „p1won“ als Kanal, true als Wert, Prozess Quit: Quit Game

Darstellung in Psedudo-Code	Darstellung in Pi-Kalkül
`(Process other)` `other.p1won(true);`	$\overline{p1won}$<true>.Quit

Nebenläufigkeit P | Q

Beispiel: Eingabe und Ausgabe von vorher nebenläufig

p1won(boolean).Print | $\overline{p1won}$<true>.Quit → Print| Quit (print „P1 won“ und Quit Game)

Summation P + Q

Erklärung: Die Summation stellt nichtdeterminus dar.

Beispiel:

P = p1won(boolean).Print1

Q = p2won(boolean).Print2

Output = $\overline{p1won}$<true>.Quit

Output | P + Q

$\overline{p1won}$<true>.Quit | (p1won(boolean).Print1 + p2won(boolean).Print2) → Print1 | Quit

Replikation !P

unendlich viele Ausführungen von P → !P | P | P | P … Beispiel automatische Ressourcenerzeugung in einem Spiel: je Tick löse Prozess „Ressourcen erhöhen“ aus

es geht aber auch: !c(x).P → P wird nur repliziert wenn auf Kanal c empfangen wird $\bar{c}$<y>.P | !c(x).P → !c(x).P | P

Beispiel: Spiel: auf Knopfdruck wird neue Einheit in Fabrik hergestellt

„privater Kanal / lokale Namen“ νx.P

lokaler Kanal x ist nur in P bekannt

Beispiel:

Rundenbasierter Spielablauf von zwei Spielern.

!((ν einheitbauen)(spieler1(_).$\overline{einheitbauen}$<zwei>.spieler2<_> | einheitbauen(anzahl))) |

!((ν einheitbauen)(spieler2(_).$einheitbauen$<zwei>.spieler1<_> | einheitbauen(anzahl)))

Den Kanal einheitbauen gibt es bei beiden Spielern, doch wird dieser nur lokal betrachtet.

Freie und Gebundene Namen

Freie Namen		**Gebundene Namen**	
fn(x(y).P)	={x}∪(fn(P)\{y})	fn(x(y).P)	={x}∪(fn(P)\{y})
fn($\bar{x}$<y>.P)	={x, y}∪fn(P)	fn($\bar{x}$<y>.P)	={x, y}∪fn(P)
fn(P_1 | P_2)	=fn(P_1)∪fn(P_2)	fn(P_1 | P_2)	=fn(P_1)∪fn(P_2)
fn(0)	=∅	fn(0)	=∅
fn(νx.P)	=fn(P)\{x}	fn(νx.P)	=fn(P)\{x}
fn(!P)	=fn(P)	fn(!P)	=fnP

3. Unterschied Synchron vs asynchron:

Bei dem asynchronen Pi-Kalkül darf nach dem Output nur der Nullprozess folgen.

also gilt: $\bar{c}<y>.P$ ist nicht erlaubt
$\bar{c}<y>.0$ ist erlaubt

4. Weitere interessante Beispiele

Mehrere Eingabe- und Ausgabekanäle mit gleichen Namen

$\bar{v}<x>.0 \mid v(a).P \mid v(b).Q \rightarrow v(a).P \mid Q$ oder $P \mid v(b).Q$

Wenn mehrere Prozesse den gleichen Namen für Input und Output Kanäle verwenden, gilt Nichtdeterminismus.

Den Input-Kanal nutzen, um einen neuen Kommunikationsweg zu öffnen

$v(y).y<x>.0 \mid a(z).P \mid v<a>.0 \rightarrow a<x>.0 \mid a(z).P \rightarrow P[x/z]$

Beim Empfang der Nachricht auf v, wird y substituiert durch den empfangenen Namen a.

Synchronisation im asynchronen Pi-Kalkül

$\bar{x}<z>.P \mid x(y).Q \rightarrow$

$\bar{x}<z>.0 \mid u(\_).P \mid x(y).(u<\_>.0 \mid Q) \rightarrow$

$u(\_).P \mid \bar{u}<\_>.0 \mid Q[z/y] \rightarrow$

$P \mid Q[z/y]$

Bereich für Lokale Namen erweitern (Extrusion)

$x(y).P \mid (v\ z)(\ \bar{x}<z>.Q) \rightarrow (v\ z)(P[z/y] \mid Q)$

Bessere Synchronisation im asynchronen Pi-Kalkül durch Extrusion

$\bar{x}<z>.P \mid x(y).Q \rightarrow$

$\bar{x}<z>.P \Rightarrow (v\ a)(\ \bar{x}<a>.0 \mid a(c).(\ \bar{c}<z>.0 \mid P\ ))$ mit a,c nicht in fn(P)

$x(y).Q \Rightarrow (v\ b)(x(e).e<b>.0 \mid b(y).Q)$ mit b,e nicht in fn(Q)

$(v\ a)(\ \bar{x}<a>.0 \mid a(c).(\ \bar{c}<z>.0 \mid P\ )) \mid (v\ b)(x(e).\ \bar{e}<b>.0 \mid b(y).Q) \rightarrow$

$(v\ a)(a(c).(\bar{c}<z>.0 \mid P\ ) \mid (v\ b)\bar{a}<b>.0 \mid b(y).Q)) \rightarrow$ //Bereich des lokalen Namen a wurde erweitert,
neuer Kommunikationsweg

$(v\ a)\ (v\ b)(\bar{b}<z>.0 \mid P\ \mid b(y).Q) \rightarrow$ //Bereich des lokalen Namen b wurde erweitert,
neuer Kommunikationsweg

$(v\ a)\ (v\ b)\ (P \mid Q[z/y]) \rightarrow$

$P \mid Q[z/y]$