1 Parallele disjunkte Programme

1.1 Syntax und Semantik

Programm S_1, S_2 disjunkt:
$change(S_1) \cap var(S_2) = \emptyset$
$\wedge change(S_2) \cap var(S_1) = \emptyset$

S und ein Ausdruck p
eine Zusicherung t disjunkt:
$change(S) \cap var(p) = \emptyset$
$change(S) \cap var(t) = \emptyset$

Beispiel

$$\begin{array}{ll} disjunkt: & S_1 \equiv x := z, S_2 \equiv y := z \\ nicht\ disjunkt: & S_1 \equiv x := y, S_2 \equiv y := z \end{array}$$

Syntax und Semantik deterministischer Programme plus:

disjunkte parallele Komposition
$S ::= [S_1\|\ldots\|S_n]$
$S_1, \ldots, S_n:\ disjunkt,\ deterministisch$

Transitionsregel:

$$\frac{< S_i, \sigma > \to < T_i, \tau >}{< [\ldots\|S_i\|\ldots], \sigma > \to < [\ldots\|T_i\|\ldots], \tau >}$$

"interleaving concurrency modeling"
Beispiel:

$$\begin{array}{l} < [x := 1\|y := 2\|z := 3], \sigma > \\ \to < [E\|y := 2\|z := 3], \sigma[x := 1] > \\ \to < [E\|E\|z := 3], \sigma[x := 1][y := 2] > \\ \to < [E\|E\|E], \sigma[x := 1][y := 2][z := 3] > \end{array}$$

1.1.1 Termination

$$[E\|\ldots\|E] \equiv ES\ terminiert \leftrightarrow S_1 \ldots S_n\ terminiert$$

1.1.2 Kein Blockieren

$$\begin{array}{l} \forall < S, \sigma >, \\ S \neq E,\ \sigma\ ein\ gültiger\ Zustand: \\ \exists < T, \tau >: < S, \sigma > \to < T, \tau > \end{array}$$

1.1.3 Menge der Endzustände

$$\begin{array}{lll} \mathcal{M}[\![S]\!](\sigma) & = & \{\tau | < S, \sigma > \to^* < E, \tau >\} \\ \mathcal{M}_{tot}[\![S]\!](\sigma) & = & \mathcal{M}[\![S]\!](\sigma) \cup \\ & & \{\bot | S\ divergiert\ in\ \sigma\} \end{array}$$

1.1.4 Korrektheit

$$\models_{part} \{p\}S\{q\} \ wenn \ \mathcal{M}_{part}[\![S]\!]([\![p]\!]) \subseteq [\![q]\!]$$

1.1.5 Determinismus

Disjunktivität verbietet:

- Beeinflussung des Kontrollfluß
- Veränderung der Daten

zwischen den parallelen Programmen. Folge:

Alle Durchgänge eines disjunkten parallelen Programmes produzieren vom selben Initialzustand aus denselben Zustand:

$$|\mathcal{M}_{tot}[\![S]\!](\sigma)| = 1$$

1.1.6 Sequenzialisierung

Daraus folgt die Möglichkeit der Sequenzialisierung:

$$\mathcal{M}[\![[S_1\|\ldots\|S_n]]\!] = \mathcal{M}[\![S_1;\ldots;S_n]\!]$$
$$\mathcal{M}_{tot}[\![[S_1\|\ldots\|S_n]]\!] = \mathcal{M}_{tot}[\![S_1;\ldots;S_n]\!]$$

1.2 Beweis System

`AXIOM 1: SKIP`

$$\{p\}skip\{p\}$$

`AXIOM 2: ZUWEISUNG`

$$\{p[u:=t]\}u:=t\{p\}$$

`REGEL 3: KOMPOSITION`

$$\frac{\{p\}S_1\{r\},\{r\}S_2\{q\}}{\{p\}S_1;S_2\{q\}}$$

`REGEL 4: BEDINGUNG`

$$\frac{\{p\wedge B\}S_1\{q\},\{p\wedge\neg B\}S_2\{q\}}{\{p\}\ \text{ if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q\}}$$

`REGEL 5: SCHLEIFE`^{part}

$$\frac{\{p\wedge B\}S\{p\}}{\{p\}\ \text{ while } B \text{ do } S \text{ od } \{p\wedge\neg B\}}$$

`REGEL 6: KONSEQUENZ`

$$\frac{p\rightarrow p_1,\{p_1\}S\{q_1\},q_1\rightarrow q}{\{p\}S\{q\}}$$

`REGEL 7: SCHLEIFE II`^{tot}

$$\frac{\begin{array}{l}\{p\wedge B\}\ S\ \{p\},\\ \{p\wedge B\wedge t=z\}\ S\ \{t<z\},\\ p\rightarrow t\geq 0\end{array}}{\{p\} \text{ while } B \text{ do } S \text{ od}\{p\wedge\neg B\}}$$

Hilfsregeln:

`AXIOM A1: INVARIANZ`^{part}

$$\{p\}S\{p\},$$

wobei $free(p) \cap change(S) = \emptyset$.

`REGEL A2: DISJUNKTION`

$$\frac{\{p\}S\{q\}, \{r\}S\{q\}}{\{p \vee r\}S\{q\}}$$

`REGEL A3: KONJUNKTION`

$$\frac{\{p_1\}S\{q_1\}, \{p_2\}S\{q_2\}}{\{p_1 \wedge p_2\}S\{q_1 \wedge q_2\}}$$

`REGEL A4:` $\exists$`-INTRODUCTION`

$$\frac{\{p\}S\{q\}}{\{\exists x : p\}S\{q\}},$$

wobei x nicht in S oder $free(q)$ vorkommt.

`REGEL A5: INVARIANZ`

$$\frac{\{r\}S\{q\}}{\{p \wedge r\}S\{p \wedge q\}},$$

wobei $free(p) \cap change(S) = \emptyset$.

`REGEL 15: SEQUENZIALISIERUNG`

$$\frac{\{p\}S_1; \ldots ; S_n\{q\}}{\{p\}[S_1 \| \ldots \| S_n]\{q\}}$$

`REGEL 16: DISJUNKTE PARALLELITÄT`

$$\frac{\{p_i\}S_i\{q_i\}, i \in \{1, \ldots, n\}}{\left\{\bigwedge_{i=1}^{n} p_i\right\}[S_1 \| \ldots \| S_n]\left\{\bigwedge_{i=1}^{n} q_i\right\}}$$

`REGEL 17: HILFSVARIABLEN`

Kommt eine Variable x in einem Programm nur in der Form $x := t$ vor, ist x eine Hilfsvariable.

$$\frac{\{p\}S\{q\}}{\{p\}S_0\{q\}},$$

wobei S_0 aus S entsteht, wenn alle Zuweisungen an Hilfsvariablen aus S entfernt werden.

1.3 Fairness

Der Lauf eines nicht deterministischen oder parallelen Programmes:

$$E_0, i_0) \ldots (E_j, i_j) \ldots ,$$

wobei E_i die geordnete Menge der jeweils bereiten Komponenten ist und i_j die jeweilige Selektion (des Schedulers). Der Lauf ist fair, wenn gilt:

$$\forall (1 \leq i \leq n) : (\overset{\infty}{\exists}\ j \in N_0 : i \in E_j \rightarrow \overset{\infty}{\exists}\ j \in N_0 : i = i_j)$$

Für parallele Programm bedeutet bereit nicht terminiert. Daher:

$$\forall(1 \leq i \leq n) : \forall j \in N_0 : (i \in E_j \rightarrow \exists k \geq j : i = i_k).$$

Faire parallele Semantik wird definiert als:

$$\begin{aligned} M_{fair}[\![S]\!](\sigma) = & \quad \{\tau | < S, \sigma > \rightarrow^* < E, \tau >\} \\ \cup & \quad \{\perp | S \text{ can diverge from } \sigma \text{ in a fair computation}\} \end{aligned}$$

Für disjunkte, parallele Programm gilt $M_{tot} = M_{fair}$.

2 Parallele Programme mit gemeinsamen Variablen

2.1 Syntax und Semantik

2.1.1 Syntax

- parallele Komposition ohne Disjunktivität
- atomare Regionen
- "interleaving concurrency modeling"

Syntax und Semantik deterministischer Programme plus:

atomare Regionen
$S ::= \langle S_0 \rangle$
$S_0 :$ *deterministisch*,
schleifenfrei,
enthält keine atomare Region

parallele Komposition
$S ::= [S_1 \| \ldots \| S_n]$
$S_1, \ldots, S_n :$ *deterministisch*

Abwechselnde Ausführung der atomaren Schritte:

atomare Region
Boolescher Ausdruck
Zuweisung
skip

Ein normales Subprogramm von S darf nicht Anteile eines atomaren Schrittes von S enthalten.

$\{p\}\ S^*\ \{q\}$ ist Standard Beweis Skizze, wenn jedem normalen Subprogramm T genau eine Zusicherung $pre(T)$ vorangeht.

2.1.2 Transitionsregel

$$\frac{< S, \sigma > \to^* < E, \tau >}{< \langle S \rangle, \sigma > \to < E, \tau >}$$

$$\frac{< S_i, \sigma > \to < T_i, \tau >}{< [\ldots \| S_i \| \ldots], \sigma > \to < [\ldots \| T_i \| \ldots], \tau >}$$

2.1.3 Kein Blockieren

2.1.4 Begrenzter Nichtdeterminismus

$\mathcal{M}_{tot}[\![S]\!](\sigma)$ ist endlich oder enthält $\perp$.

2.1.5 Keine Kompositionalität

Es gibt keine Operation *op*, die aus den Endzuständen der Komponenten die Enzustände der Komposition bildet:

$$\mathcal{M}_{tot}[\![[S_1 \| \ldots \| S_n]]\!] = op(\mathcal{M}[\![S_1]\!], \ldots, [\![S_n]\!])$$

2.2 Beweis System

2.2.1 Beweis System PSV für partielle Korrektheit

Regeln 1-6,17 und A1-A5 und folgende.

Beweis System ohne Regel für Hilfsvariablen nicht vollständig!

`REGEL 18: ATOMARE REGION`

$$\frac{\{p\}\ S\ \{q\}}{\{p\}\ \langle S \rangle\ \{q\}}$$

2.2.2 Interferenz Freiheit

Ein Programm R interferiert nicht mit der Standard Beweis Skizze $\{p\}\ S^*\ \{q\}$, wenn für jede Zusicherung r gilt: $\{r \wedge pre(R)\}\ R\ \{r\}$.

Die Standard Beweis Skizzen $\{p_i\}\ S_i^*\ \{q_i\},\ i \in \{1, \ldots, n\}$ interferieren nicht, wenn kein normales Subprogramm von S_j mit der Standard Beweis Skizze $\{p_i\}\ S_i^*\ \{q_i\}$ interferiert, $i \neq j$.

`REGEL 19: PARALLELISMUS MIT GEMEINSAMEN VARIABLEN`

$$\frac{\{p_i\}\ S_i^*\ \{q_i\},\ i \in \{1, \ldots, n\}}{\{\bigwedge_{i=1}^n p_i\}\ [S_1 \| \ldots \| S_n]\ \{\bigwedge_{i=1}^n q_i\}},$$

wobei $\{p_i\}\ S_i^*\ \{q_i\},\ i \in \{1, \ldots, n\}$ nicht interferieren.

2.2.3 Totale Korrekheit

Forderungen an die gebundene Funktion t bei Schleifen: (LOOP II) S = Schleifenkörper

1. alle Zuweisungen und atomare Regionen in S vermindern oder erhalten t.

2. für jeden möglichen Pfad durch S wird durch mind. eine Zuweisung oder atomare Region t vermindert.

Dabei ist die Menge der Pfade durch S $path(S)$ definiert als:

- $path(skip) = \{\varepsilon\}$
- $path(u := t) = \{u := t\}$
- $path(< S >) = \{< S >\}$
- $path(S_1, S_2) = path(S_1); path(S_2)$
- $path($ **if** B **then** S_1 **else** S_2 **fi** $) = path(S_1) \cup path(S_2)$
- bold $path($ while B do S od $) = \{\varepsilon\}$

$$\Pi_1; \Pi_2 = \{\pi_1; \pi_2 | \pi_1 1 \in \Pi_1 \wedge \pi_2 \in \Pi_2\} \pi; \varepsilon = \varepsilon; \pi = \pi$$

Beweis Skizze Format Regel für Schleifen:

1. $\{p \wedge B\}$ S^* $\{p\}$ ist Standard
2. $\{pre(R) \wedge t = z\}$ R $\{t \leq z\}$ für jede normale Zuweisung oder atomare Region R in S
3. für jeden Pfad π existiert eine normale Zuweisung oder atomare Region R in π, so daß $\{pre(R) \wedge t = z\}$ R $\{t < z\}$
4. $p \rightarrow t \geq 0$

$$\{\mathbf{inv} : p\}\{\mathbf{bd} : t\} \text{ } \mathbf{while} \text{ } B \text{ } \mathbf{do} \text{ } \{p \wedge B\} \text{ } S^*\{p\} \text{ } \mathbf{od} \text{ } \{p \wedge \neg B\}$$

Die Annahme von Fairness beeinflusst PSV, aber nicht DPP !

3 Synchonisierte Programme

3.1 Syntax

Deterministische Programmsyntax plus:

- Parallele Komposition
$$S ::= [S_1 \| \dots \| S_n]$$
S_1 bis S_n sequentielle Komponenten.

- Synchronisation
$$S ::= \mathbf{await} \text{ } B \text{ } \mathbf{then} \text{ } S_0 \text{ } \mathbf{end}$$
S_0 schleifenfrei und ohne **await** Anweisungen. **await** Anweisungen dürfen nur innerhalb einer parallelen Komposition auftreten.

3.2 Semantik

3.2.1 Transition

- Parallele Komposition

$$\frac{< S_i, \sigma > \to < T_i, \tau >}{< [\ldots \| S_i \| \ldots], \sigma > \to < [\ldots \| T_i \| \ldots], \tau >}$$

- Synchronisation

$$\frac{< \ S, \sigma \ \ > \to^* < \ E, \tau \ \ >}{< \ \textbf{await}\ B\ \textbf{then}\ S\ \textbf{end}, \sigma \ \ > \to < \ E, \tau \ \ >}$$

wenn $\sigma \models B$. Wenn B $false$ ist, blockiert die Anweisung. S wird atomar ausgeführt. Die Synchronisation ist also eine bedingte atomare Region.

3.2.2 *deadlock*

Ein *deadlock* ist vorhanden, wenn alle Komponenten blockiert oder terminiert sind.

3.2.3 Endzustand

- Semantik für partielle Korrektheit

$$\mathcal{M}[\![S]\!](\sigma) = \{\tau | < \ S, \sigma \ \ > \to^* < \ E, \tau \ \ >\}$$

- Semantik für schwache totale Korrektheit

$$\mathcal{M}_{wtot}[\![S]\!](\sigma) = \mathcal{M}[\![S]\!](\sigma) \cup \{\bot | S \text{ kann divergieren ab } \sigma\}$$

- Semantik für totale Korrektheit

$$\mathcal{M}_{tot}[\![S]\!](\sigma) = \mathcal{M}_{wtot}[\![S]\!](\sigma) \cup \{\Delta | S \text{ kann blockieren ab } \sigma\}$$

3.3 Beweis Theorie

3.3.1 Beweis Regel

REGEL 20: SYNCHRONISATION

$$\frac{\{p \wedge B\}\ S\ \{q\}}{\{p\}\ \textbf{await}\ B\ \textbf{then}\ S\ \textbf{end}\ \{q\}}$$

3.3.2 Beweis Skizze

$$\frac{\{p \wedge B\}\ S^*\ \{q\}}{\{p\}\ \textbf{await}\ B\ \textbf{then}\ \{p \wedge B\}\ S^*\ \{q\}\ \textbf{end}\ \{q\}},$$

wobei S^* eine Version mit Anmerkungen von S ist.

Korrektheit und Vollständigkeit der Beweisskizze ergibt sich aus Korrektheit und Vollständigkeit der Beweisskizzen für deterministische Programme.

Das Beweissystem besteht aus den Regeln 1 - 6, 17, 19, 20 und A1 - A5.

Eine interferenzfreie Standard Beweis Skizze

$$\{p_i\}S_i^*\{q_i\}, i \in \{1,\ldots,n\}.$$

Für einen Zustand σ, so daß $\wedge_{i=1}^n p_i$ gilt, und

$$< [S-1\|\ldots\|S_n], \sigma > \to^* < [R_1\|\ldots\|R_n], \tau >:$$

- wenn $R_j \equiv \mathbf{at}(T, S_j)$, T normales Unterprogramm von S_j, dann $\tau \models pre(T)$
- wenn $R_j \equiv E$, dann $ttau \models q_j$

Im ersten Falle sei r $pre(T)$ und im zweiten Falle q_j. Induktion über Länge der Transition: Für 0 gilt: $p_j \to r, \sigma = \tau$ also $\tau \models r$, da $\sigma \models p_j$. Bei Länge über 0 gilt für ein R....

3.3.3 Schwache totale Korrektheit

Für schwache totale Korrektheit muß bewiesen werden, daß das Programm nicht divergiert. Die Beweisverfahren und Regeln für Beweiskizzen sind wie für totale Korrektheit bei den bisherigen parallelen Programmen. *await* Anweisungen werden wie atomare Regionen behandelt: Die Definition von *path* wird um

- $path(\mathbf{await}\ B\ \mathbf{then}\ S\ \mathbf{end} = \{\mathbf{await}\ B\ \mathbf{then}\ S\ \mathbf{end}\}$

erweitert. Ansonsten erfolgt der Nachweis der Konvergenz wie oben.

3.3.4 Totale Korrektheit

Der Nachweis der totalen Korrektheit erfordert zusätzlich den Nachweis der *deadlock* Freiheit.

Dazu werden zuerst die potentiellen *deadlock* Situationen aufgezählt:

- $(R_1, \ldots, R_n)$ ist ein potenteller *deadlock*, wenn jedes R_i eine **await** Anweisung in S_i oder E ist und zumindest ein R_i eine **await** Anweisung in S_i ist.
- $\{p_i\}$ S^* $\{q_i\}$ ist eine Beweisskizze für totale Korrektheit. Zu jedem potentellen *deadlock* gibt es ein Tupel $(r_1, \ldots, r_n)$ von Zusicherungen:
  - $r_i \equiv pre(R_i) \wedge \neg B$ if $R_i \equiv \mathbf{await}\ B\ \mathbf{then}\ S\ \mathbf{end}$
  - $r_i \equiv q_i$ if $R_i \equiv E$

Dann ergibt sich folgende

`RULE 21: DEADLOCK FREIER PARALLELISMUS`

(1) $\{p_i\}$ S^* $\{q_i\}$ interferenzfreie Beweisskizze für schwach totale Korrektheit
(2) $\forall(R_1, \ldots, R_n) : \neg \wedge_{i=1}^n r_i$

$\{\wedge_{i=1}^n p_i\}[S_1\|\ldots\|S_n]\{\wedge_{i=1}^n q_i\}$

Beweis System für totale Korrektheit von synchronisierten parallelen Programmen: Regel 1-5,7,17,20,21 und A2-A5.

Beweis Skizzen werden entsprechend angefertigt.
Beispiel:

$$\{x = 0\}\ [\textbf{await}\ x = 1\ \textbf{then}\ \textit{skip}\ \textbf{end} \| x := 1]\ \{x = 1\}$$
$$\{x = 0 \vee x = 1\}\ [\textbf{await}\ x = 1\ \textbf{then}\ \textit{skip}\ \textbf{end}\ \{x = 1\}$$
$$\{x = 0\}\ x := 1\ \{x = 1\}$$

Potentielle *deadlocks*:

$$(\textbf{await}\ x = 1\ \textbf{then}\ \textit{skip}\ \textbf{end}, E)$$

mit den Zusicherungen:

$$((x = 0 \vee x = 1) \wedge x \neq 1, x = 1)$$

Kein *deadlock* möglich !

3.4 Fallstudie: Gegenseitiger Ausschluß

3.4.1 Problemstellung

Es soll verifiziert werden, daß das folgende parallele Programm den gegenseitigen Ausschluß bestimmter Regionen seiner Komponenten garantiert. Die Komponenten seien wie folgt gebaut:

$$\begin{array}{ll} S_i \equiv & \textbf{while true do} \\ & \quad NC_i; \\ & \quad ACQ_i; \\ & \quad CS_i; \\ & \quad REL_i \\ & \textbf{od} \end{array}$$

Die Programmteile haben folgende Bedeutung:

Programm	Bedeutung
NC_i	non-critical section
ACQ_i	acquire protocl
CS_i	critical section
REL_i	release protocol

Das Gesamtprogramm soll die Form

$$S \equiv INIT;\ [S_1 \| \ldots \| S_n]$$

haben. Formalisierte Bedingungen:

1. Gegenseitiger Ausschluß

$$\forall i, j \in \{1, \ldots, n\} : \neg(pre(CS_i) \wedge pre(CS_j))$$

2. *deadlock* Freiheit

3.4.2 Lösung mit binärer Semaphore

- Initialisierung $sem := \mathbf{true}$
- P Operation $P(sem) \equiv \mathbf{await}\ sem\ \mathbf{then}\ sem := \mathbf{false\ end}$
- V Operation $P(sem) \equiv sem := \mathbf{true}$

$$MUTEX \equiv out := \mathbf{true}; [S_1 \| \ldots \| S_n]$$

$S_i \equiv$ **while true do**
NC_i;
$P(out)$;
CS_i;
$V(out)$
od

$\{\mathbf{inv : true}\}$
while true do
$\{\mathbf{true}\}$
NC_i;
$\{\mathbf{true}\}$
await out **then** $out := \mathbf{false}; who := i$ **end**;
$\{\neg out \wedge who = i\}$
CS_i; $\{\neg out \wedge who = i\}$
out :=**true**
od
$\{\mathbf{false}\}$

3.4.3 Beweis der Interferenzfreiheit

1. $\{\neg out \wedge who = i \wedge out\}$
$\{\mathbf{false}\}$
$out := \mathbf{false}; who := j$
$\{\mathbf{false}\}$
$\{\neg out \wedge who = i\}$

Nach Regel 20:

$\{\neg out \wedge who = i \wedge \{\mathbf{true}\}$
await out **then** $out := \mathbf{false}; who := j$ **end**
$\{\neg out \wedge who = i\}$

2. $\{\neg out \wedge who = i \wedge \neg out \wedge who = j\}$
$\{\mathbf{false}\}$
$out := \mathbf{true}$
$\{\mathbf{false}\}$
$\{\neg out \wedge who = i\}$

3.4.4 Beweis des gegenseitigen Ausschluß

$$pre(CS_i) \wedge pre(CS_j) \rightarrow who = i \wedge who = j \rightarrow \neg(pre(CS_I) \wedge pre(CS_j))$$

3.4.5 Beweis von *deadlock* Freiheit

$$
\begin{array}{l}
\{\textbf{inv}: who \neq i \wedge I\} \\
\textbf{while true do} \\
\quad \{who \neq i \wedge I\} \\
\quad NC_i; \\
\quad \{who \neq i \wedge I\} \\
\quad \textbf{await}\ out\ \textbf{then}\ out := \textbf{false}; who := i\ \textbf{end}; \\
\quad \{\neg out \wedge who = i\} \\
\quad CS_i;\ \{\neg out \wedge who = i\} \\
\quad \langle out := \textbf{true}; who := 0\rangle \\
\textbf{od} \\
\{\textbf{false}\}
\end{array}
$$

mit

$$I \equiv (\vee_{j=0}^{n} who = j) \wedge (who = 0 \leftrightarrow out)$$

Für einen potentiellen *deadlock* muß man annehmen, daß alle Komponenten bei der **await** Anweisung stehen, da sie auf keinen Fall terminieren. Das entsprechende Tupel von gültigen Zusicherungen $(r_1, \ldots, r_n)$ ist gegeben durch:

$$
\begin{array}{rc}
 & r_i \equiv who \neq i \wedge I \wedge \neg out \\
\rightarrow & (\wedge_{i=1}^{n} r_i) \rightarrow who = 0 \wedge (who = 0 \leftrightarrow out) \wedge \neg out \\
\rightarrow & \neg \wedge_{i=1}^{n} r_i)
\end{array}
$$

4 Verteilte Programme

Syntax wie CSP plus *guarded commands*, Beweis durch FDR.