

1 Definitionen

Stand: 5. Juli 2017

Kombination aus [BV15] und [Sch16] mit Einflüssen von [BFLV16]:

Definition 1.1 (*Modal Error-I/O-Transitionssystem*). Ein Modal Error-I/O-Transitionssystem (MEIO) ist ein Tupel $(P, I, O, \longrightarrow, \dashrightarrow, p_0, E)$ mit:

- P : Menge der Zustände,
- $p_0 \in P$: Startzustand,
- I, O : disjunkte Mengen der (sichtbaren) Input- und Output-Aktionen,
- $\longrightarrow \subseteq P \times \Sigma_\tau \times P$: must-Transitions-Relation,
- $\dashrightarrow \subseteq P \times \Sigma_\tau \times P$: may-Transitions-Relation,
- $E \subseteq P$: Menge der Fehler-Zustände.

Es wird vorausgesetzt, dass $\longrightarrow \subseteq \dashrightarrow$ (syntaktische Konsistenz) gilt.

Das Alphabet bzw. die Aktionsmenge eines MEIO ist $\Sigma = I \cup O$. Die interne Aktion τ ist nicht in Σ enthalten. Jedoch gilt $\Sigma_\tau := \Sigma \cup \{\tau\}$. Die Signatur eines MEIOs entspricht $\text{Sig}(P) = (I, O)$.

Falls $\longrightarrow = \dashrightarrow$ gilt, wird P auch Implementierung genannt.

Implementierungen entsprechen den in [Sch16] behandelten EIOs.

Must-Transitionen sind Transitionen, die von einer Verfeinerung implementiert werden müssen. Die may-Transitionen sind hingegen die zulässigen Transitionen für eine Verfeinerung.

MEIOs werden in dieser Arbeit durch ihre Zustandsmenge (z.B. P) identifiziert und falls notwendig werden damit auch die Komponenten indiziert (z.B. I_P anstatt I). Falls der MEIO selbst bereits einen Index hat (z.B. P_1) kann an der Komponente die Zustandsmenge als Index wegfallen und nur noch der Index des gesamten Automaten verwendet werden (z.B. I_1 anstatt I_{P_1}). Zusätzlich stehen i, o, a, ω und α für Buchstaben aus den Alphabeten $I, O, \Sigma, O \cup \{\tau\}$ und Σ_τ .

Es wird die Notation $p \xrightarrow{\alpha} p'$ für $(p, \alpha, p') \in \dashrightarrow$ und $p \xrightarrow{\alpha}$ für $\exists p' : (p, \alpha, p') \in \dashrightarrow$ verwendet. Dies kann entsprechend auf Buchstaben-Sequenzen $w \in \Sigma_\tau^*$ erweitert werden zu $p \xrightarrow{w} p'$ ($p \xrightarrow{w}$) steht für die Existenz eines Laufes $p \xrightarrow{\alpha_1} p_1 \xrightarrow{\alpha_2} \dots p_{n-1} \xrightarrow{\alpha_n} p'$ ($p \xrightarrow{\alpha_1} p_1 \xrightarrow{\alpha_2} \dots p_{n-1} \xrightarrow{\alpha_n}$) mit $w = \alpha_1 \dots \alpha_n$.

1 Definitionen

Desweiteren soll $w|_B$ die Aktions-Sequenz bezeichnen, die man erhält, wenn man aus w alle Aktionen löscht, die nicht in $B \subseteq \Sigma$ enthalten sind. \hat{w} steht für $w|_\Sigma$. Es wir $p \stackrel{w}{\Rightarrow} p'$ für ein $w \in \Sigma^*$ geschrieben, falls $\exists w' \in \Sigma_\tau^* : \hat{w}' = w \wedge p \xrightarrow{w'} p'$, und $p \stackrel{w}{\Rightarrow} p'$ für ein beliebiges p' gilt. Falls $p_0 \stackrel{w}{\Rightarrow} p$ gilt, dann wird w *Trace* genannt und p ist ein *erreichbarer Zustand*.

Analog zu $\xrightarrow{\cdot}$ und \Rightarrow werden \longrightarrow und \Longrightarrow für die entsprechenden Relationen der must-Transition verwendet.

Outputs und die interne Aktion werden *lokale Aktionen* genannt, da sie lokal vom ausführenden MEIO kontrolliert sind. Um eine Erleichterung der Notation zu erhalten, soll gelten, dass $p \xrightarrow{a} p'$ und $p \xrightarrow{a!} p'$ für $\nexists p' : p \xrightarrow{a} p'$ und $\nexists p' : p \xrightarrow{a!} p'$ stehen soll. In Graphiken wird eine Aktion a als $a?$ notiert, falls $a \in I$ und $a!$, falls $a \in O$. Must-Transitionen (may-Transitionen) werden als durchgezogener Pfeil gezeichnet (gestrichelter Pfeil). Entsprechend der syntaktischen Konsistenz repräsentiert jede gezeichnete must-Transition auch gleichzeitig die zugrundeliegende may-Transitionen.

Definition 1.2 (Parallelkomposition). Zwei MEIOs $P_1 = (P_1, I_1, O_1, \longrightarrow_1, \xrightarrow{\cdot}_1, p_{01}, E_1)$ und $P_2 = (P_2, I_2, O_2, \longrightarrow_2, \xrightarrow{\cdot}_2, p_{02}, E_2)$ sind komponierbar, falls $O_1 \cap O_2 = \emptyset$. Für solche MEIOs ist die Parallelkomposition $P_{12} := P_1 \parallel P_2 = ((P_1 \times P_2), I, O, \longrightarrow_{12}, \xrightarrow{\cdot}_{12}, (p_{01}, p_{02}), E)$ definiert mit: TODO: erzwungenen Zeilenumbrüche kontrollieren

- $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2),$
- $O = (O_1 \cup O_2),$
- $\longrightarrow_{12} = \left\{ ((p_1, p_2), \alpha, (p'_1, p'_2)) \mid p_1 \xrightarrow{\alpha}_1 p'_1, \alpha \in \Sigma_\tau \setminus \text{Synch}(P_1, P_2) \right\} \\ \cup \left\{ ((p_1, p_2), \alpha, (p_1, p'_2)) \mid p_2 \xrightarrow{\alpha}_2 p'_2, \alpha \in \Sigma_\tau \setminus \text{Synch}(P_1, P_2) \right\} \\ \cup \left\{ ((p_1, p_2), \alpha, (p'_1, p'_2)) \mid p_1 \xrightarrow{\alpha}_1 p'_1, p_2 \xrightarrow{\alpha}_2 p'_2, \alpha \in \text{Synch}(P_1, P_2) \right\}$
- $\xrightarrow{\cdot}_{12} = \left\{ ((p_1, p_2), \alpha, (p'_1, p'_2)) \mid p_1 \xrightarrow{\alpha}_1 p'_1, \alpha \in \Sigma_\tau \setminus \text{Synch}(P_1, P_2) \right\} \\ \cup \left\{ ((p_1, p_2), \alpha, (p_1, p'_2)) \mid p_2 \xrightarrow{\alpha}_2 p'_2, \alpha \in \Sigma_\tau \setminus \text{Synch}(P_1, P_2) \right\} \\ \cup \left\{ ((p_1, p_2), \alpha, (p'_1, p'_2)) \mid p_1 \xrightarrow{\alpha}_1 p'_1, p_2 \xrightarrow{\alpha}_2 p'_2, \alpha \in \text{Synch}(P_1, P_2) \right\}$
- $E = (P_1 \times E_2) \cup (E_1 \times P_2) \quad \text{geerbte Kommunikationsfehler} \\ \left. \begin{array}{l} \cup \left\{ (p_1, p_2) \mid \exists a \in O_1 \cap I_2 : p_1 \xrightarrow{a} p_2 \xrightarrow{a!} \right\} \\ \cup \left\{ (p_1, p_2) \mid \exists a \in I_1 \cap O_2 : p_1 \xrightarrow{a!} p_2 \xrightarrow{a} \right\} \end{array} \right\} \quad \text{neue Kommunikationsfehler}$

Dabei bezeichnet $\text{Synch}(P_1, P_2) = (I_1 \cap O_2) \cup (O_1 \cap I_2) \cup (I_1 \cap I_2)$ die Menge der zu synchronisierenden Aktionen. Die synchronisierten Aktionen werden als Output bzw. Input der Komposition beibehalten.

Ein neuer Kommunikationsfehler entsteht, wenn einer der MEIOs die Möglichkeit für einen Output hat (may-Transition) und der andere MEIO der passenden Input nicht

erzwingt (keine must-Transition vorhanden). Es muss also in möglichen Implementierungen nicht wirklich zu diesem Kommunikationsfehler kommen, da die Output-Transition nicht zwingendermaßen implementiert werden muss.

Wie bereits in [Sch16] kann es durch die Synchronisation von Inputs zu keinen neuen Kommunikationsfehler kommen, da dies in beiden Automaten keine lokal kontrollierte Aktion ist. Falls jedoch nur einer der Automaten die Möglichkeit für einen Input hat, der synchronisiert wird, besteht diese Möglichkeit in der Parallelkomposition nicht mehr. Es kann also in der Kommunikation mit einem weiteren MEIO dort zu einen neuen Kommunikationsfehler kommen.

Definition 1.3 ((starke) Simulation). Eine (starke) alternierende Simulation ist eine Relation $R \subseteq P \times Q$ auf zwei MEIOs P und Q , falls für alle $(p, q) \in R$ gilt:

1. $q \xrightarrow{\alpha} q'$ impliziert $p \xrightarrow{\alpha} p'$ für ein p' mit pRq' ,
2. $p \xrightarrow{-\alpha} p'$ impliziert $q \xrightarrow{-\alpha} q'$ für ein q' mit pRq' ,

Die Vereinigung \sqsubseteq_{as} aller dieser Relationen wird als (starke) as-Verfeinerung(-s Relation) (auch modal Verfeinerung) bezeichnet. Es wird $P \sqsubseteq_{\text{as}} Q$ geschrieben, falls $p_0 \sqsubseteq_{\text{as}} q_0$ gilt, und P as-verfeinert Q (stark) oder P ist eine (starke) as-Verfeinerung von Q .

Für einen MEIO Q und eine Implementierung P mit $P \sqsubseteq_{\text{as}} Q$, ist P eine as-Implementierung von Q und es wird $\text{as-impl}(Q)$ für die Menge aller as-Implementierungen von Q verwendet.

Definition 1.4 (schwache Simulation). Eine schwache alternierende Simulation ist eine Relation $R \subseteq P \times Q$ auf zwei MEIOs P und Q , falls für alle $(p, q) \in R$ gilt:

1. $q \xrightarrow{i} q'$ impliziert $p \xrightarrow{i} \xRightarrow{\varepsilon} p'$ für ein p' mit pRq' ,
2. $q \xrightarrow{\omega} q'$ impliziert $p \xRightarrow{\hat{\omega}} p'$ für ein p' mit pRq' ,
3. $p \xrightarrow{-i} p'$ impliziert $q \xrightarrow{-i} \xRightarrow{\varepsilon} q'$ für ein q' mit pRq' ,
4. $p \xrightarrow{-\omega} p'$ impliziert $q \xRightarrow{\hat{\omega}} q'$ für ein q' mit pRq' ,

Wobei $i \in I$ und $\omega \in O \cup \{\tau\}$.

Analog zur starken alternierenden Simulation, wird hier $\sqsubseteq_{\text{w-as}}$ als Relationssymbol verwendet und man kann auch entsprechend schwache as-Verfeinerung betrachten.

Ebenso kann für einen MEIO Q und eine Implementierung P mit $P \sqsubseteq_{\text{w-as}} Q$, ist P eine w-as-Implementierung von Q und es wird $\text{w-as-impl}(Q)$ für die Menge aller w-as-Implementierungen von Q verwendet.

Die schwache Simulation erlaubt interne Aktionen beim MEIO, der die entsprechende Aktion matchen muss. Jedoch ist es zwingen notwendig, dass ein Input sofort aufgeführt wird und erst dann interne Aktionen möglich sind. Da bei Input die Reaktion auf eine Aktion ist, die die Umwelt auslöst und die nicht auf den Automaten warten kann. Output hingeben können auch verzögert werden, da die Umgebung dies dann als Input aufnimmt und für diese somit nicht lokal kontrolliert ist.

Die Parallelkomposition von Wörtern und Mengen kann aus [Sch16] übernommen werden.

Definition 1.5 (*Parallelkomposition auf Traces*).

- Für zwei Wörter $w_1 \in \Sigma_1$ und $w_2 \in \Sigma_2$ ist deren Parallelkomposition definiert als:
 $w_1 \parallel w_2 := \{w \in (\Sigma_1 \cup \Sigma_2)^* \mid w|_{\Sigma_1} = w_1 \wedge w|_{\Sigma_2} = w_2\}$.
- Für zwei Mengen von Wörtern bzw. Sprachen $W_1 \subseteq \Sigma_1^*$ und $W_2 \subseteq \Sigma_2^*$ ist deren Parallelkomposition definiert als: $W_1 \parallel W_2 := \bigcup \{w_1 \parallel w_2 \mid w_1 \in W_1 \wedge w_2 \in W_2\}$.

Ebenso können die Definitionen der Funktionen `prune` und `cont` zum Abschneiden und Verlängern von Traces aus [Sch16] übernommen werden.

Definition 1.6 (*Pruning- und Fortsetzungs-Funktion*).

- $\text{prune} : \Sigma^* \rightarrow \Sigma^*, w \mapsto u$, mit $w = uv, u = \varepsilon^1 \vee u \in \Sigma^* \cdot I$ und $v \in O^*$,
- $\text{cont} : \Sigma^* \rightarrow \mathfrak{P}(\Sigma^*)^2, w \mapsto \{wu \mid u \in \Sigma^*\}$,
- $\text{cont} : \mathfrak{P}(\Sigma^*) \rightarrow \mathfrak{P}(\Sigma^*), L \mapsto \bigcup \{\text{cont}(w) \mid w \in L\}$.

Definition 1.7 (*Sprache*). Die (maximale) Sprache eines MEIOs P ist $L(P) := \{w \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \xRightarrow{w}\}$.

Für zwei komponierbare MEIOs P_1 und P_2 gilt: $L_{12} := L(P_{12}) = L_1 \parallel L_2$.

¹ ε bezeichnet das leere Wort

² $\mathfrak{P}(M)$ bezeichnet die Potenzmenge der Menge M

2 allgemeine Folgerungen

Lemma 2.1 (*as-Implementierungen und Parallelkomposition*).

1. $P'_1 \in \text{as-impl}(P_1) \wedge P'_2 \in \text{as-impl}(P_2) \Rightarrow (P'_1 \parallel P'_2) \in \text{as-impl}(P_1 \parallel P_2)$.
2. $P' \in \text{as-impl}(P_1 \parallel P_2) \Rightarrow \exists P'_1, P'_2 : P'_1 \in \text{as-impl}(P_1) \wedge P'_2 \in \text{as-impl}(P_2) \wedge P'_1 \parallel P'_2 = P'$.

Beweis.

1. Es gelte $i \in \{1, 2\}$. Jede Transition $(\dashrightarrow'_i = \rightarrow'_i)$ in P'_i hat eine entsprechende may-Transition in P_i , da Definition 1.3 2. gilt. Aufgrund der Simulations Definition 1.3, haben P_i und P'_i die gleichen Signaturen und somit gilt $\text{Synch}(P_1, P_2) = \text{Synch}(P'_1, P'_2)$.

Daraus folgt unter Verwendung, dass die Definitionen der must- und may-Transitionen der Parallelkomposition in 1.2 analog formuliert sind, dass alle Transitionen, die in $P'_1 \parallel P'_2$ enthalten sind auch in $P_1 \parallel P_2$ als may-Transitionen vorhanden sind. Es können bei der Parallelkomposition von MEIOs, die die gleichen must- und may-Transitions Mengen haben (Implementierungen), keine may-Transitionen entstehen, zu denen es keine passende must-Transition gibt (Definition von \rightarrow_{12} und \dashrightarrow_{12} in 1.2). $P'_1 \parallel P'_2$ ist also eine Implementierung.

$P_1 \parallel P_2$ enthält nur must-Transitionen, wenn auch P_1 bzw. P_2 diese enthalten haben. P'_1 und P'_2 müssen diese must-Transitionen aufgrund von Definition 1.3 1. implementieren und somit enthält auch $P'_1 \parallel P'_2$ die entsprechenden durch 1.3 1. geforderten Transitionen, um eine as-Implementierung von $P_1 \parallel P_2$ sein zu können.
 $\Rightarrow (P'_1 \parallel P'_2) \in \text{as-impl}(P_1 \parallel P_2)$.

2. Wie im Beweis zu 1. muss jede Transition aus P' in $P_1 \parallel P_2$ als may-Transition auftauchen (Definition 1.3 2.). Die gleichen Signaturen von P_i und P'_i führen ebenso wieder zu den gleichen Synchronisationsmengen.

Falls in P' eine Transition mit einer Aktion aus Synch beschriftet ist muss diese auch als may-Transition in P_1 und P_2 vorhanden sein (Argumentation von oben und Definition 1.2). Bei Aktionen, die nicht in Synch enthalten sind, muss nur der jeweilige MEIO P_1 bzw. P_2 die Transition als may-Transition enthalten.

\Rightarrow Es kann P'_1 und P'_2 geben, die alle nötigen Transitionen implementierten, so dass $P' = P'_1 \parallel P'_2$ gilt.

Alle Transitionen, die in P' aufgrund von Definition 1.3 1. implementiert werden müssen, mussten auch bereits in P_1 bzw. P_2 als must-Transitionen vorhanden sein.

2 allgemeine Folgerungen

Da P'_1 und P'_2 as-Implementierungen von P_1 und P_2 sind, müssen diese auch die Simulations Definition (1.3) erfüllen und somit die must-Transitionen aus P_1 bzw. P_2 implementieren.

Es kann nicht passieren, dass P_1 und P_2 must-Transitionen enthalten, die P'_1 und P'_2 implementieren müssen und dann in $P'_1 \parallel P'_2$ zu einer Transition führen, die P' auf Basis der Definition 1.3 1. nicht ebenfalls implementieren muss (\rightarrow_{12} und \dashrightarrow_{12} Definitionen in 1.2).

\Rightarrow es gibt passende P'_1 und P'_2 mit $P'_1 \in \text{as-impl}(P_1) \wedge P'_2 \in \text{as-impl}(P_2) \wedge P' = P'_1 \parallel P'_2$.

□

Lemma 2.2 (*w-as-Implementierungen und Parallelkomposition*).

1. $P'_1 \in \text{w-as-impl}(P_1) \wedge P'_2 \in \text{w-as-impl}(P_2) \Rightarrow (P'_1 \parallel P'_2) \in \text{w-as-impl}(P_1 \parallel P_2)$.
2. $P' \in \text{w-as-impl}(P_1 \parallel P_2) \Rightarrow \exists P'_1, P'_2 : P'_1 \in \text{w-as-impl}(P_1) \wedge P'_2 \in \text{w-as-impl}(P_2) \wedge P'_1 \parallel P'_2 = P'$.

Beweis.

1. Es gelte $i \in \{1, 2\}$. Jede Transition ($\dashrightarrow'_i = \rightarrow'_i$) einer sichtbaren Aktion in P'_i hat eine entsprechende schwache may-Transition in P_i , da Definition 1.4 2. gilt. Aufgrund der Simulations Definition 1.4, haben P_i und P'_i die gleichen Signaturen (bezieht sich nur auf sichtbare Aktionen) und somit gilt $\text{Synch}(P_1, P_2) = \text{Synch}(P'_1, P'_2)$.

Daraus folgt unter Verwendung, dass die Definitionen der must- und may-Transitionen der Parallelkomposition in 1.2 analog formuliert sind, dass alle sichtbaren Transitionen, die in $P'_1 \parallel P'_2$ enthalten sind auch in $P_1 \parallel P_2$ als schwache may-Transitionen vorhanden sind. Hierzu ist noch anzumerken, dass die interne Aktion nie in der Parallelkomposition synchronisiert wird und somit die MEIOs diese in der Komposition jeweils für ihre Komponente alleine ausführen.

$P'_1 \parallel P'_2$ ist mit der gleichen Begründung wie im Beweis von Satz 2.1 1. eine Implementierung.

$P_1 \parallel P_2$ enthält nur must-Transitionen, wenn auch P_1 bzw. P_2 diese enthalten haben. P'_1 und P'_2 müssen diese must-Transitionen aufgrund von Definition 1.3 1. schwach implementieren, d.h. bei Inputs können danach noch interne Aktionen möglich sein und bei Outputs davor und danach und bei einen τ können beliebig viele interne Aktionen implementiert werden (auch keine). Die durch $P_1 \parallel P_2$ und 1.3 1. geforderten Implementierungen von must-Transitionen in $P'_1 \parallel P'_2$ basieren auf den must-Transitionen der einzelnen MEIOs, die schwach in deren w-as-Implementierungen implementiert wurden. Die zusätzlichen τ s, die dadurch entstehen, hindern $P'_1 \parallel P'_2$ in der Parallelkomposition nicht daran die must-Transitionen auch entsprechend schwach zu implementieren.

$\Rightarrow (P'_1 \parallel P'_2) \in \text{w-as-impl}(P_1 \parallel P_2)$.

2 allgemeine Folgerungen

2. Wie im Beweis zu 1. muss jede sichtbare Transition aus P' in $P_1 \parallel P_2$ als schwache may-Transition auftauchen (Definition 1.4 2.). Bei der Projektion auf die einzelnen Transitionssysteme muss die schwache Transition im jeweiligen Automaten erhalten bleiben. Wobei sich jedoch die Anzahl der τ s auf die beiden Systeme aufteilt. Die gleichen Signaturen von P_i und P'_i führen ebenso wieder zu den gleichen Synchronisationsmengen.

\Rightarrow Es kann P'_1 und P'_2 geben, die alle nötigen Transitionen implementierten, so dass $P' = P'_1 \parallel P'_2$ gilt.

Alle Transitionen, die in P' aufgrund von Definition 1.3 1. schwach implementiert werden müssen, mussten auch bereits in P_1 bzw. P_2 als must-Transitionen vorhanden sein. Da P'_1 und P'_2 w-as-Implementierungen von P_1 und P_2 sind, müssen diese auch die Simulations Definition (1.3) erfüllen und somit die must-Transitionen aus P_1 bzw. P_2 schwach implementieren. Hierbei muss auf die korrekte Anzahl an τ s geachtet werden, die man dann in der Parallelkomposition erhält.

Es kann nicht passieren, dass P_1 und P_2 must-Transitionen enthalten, die P'_1 und P'_2 schwach implementieren müssen und dann in $P'_1 \parallel P'_2$ zu einer mit einer sichtbaren Aktion beschrifteten Transition führen, die P' auf Basis der Definition 1.3 1. nicht ebenfalls implementieren muss (\longrightarrow_{12} und \dashrightarrow_{12} Definitionen in 1.2). Natürlich kann es durch die internen Aktionen zu Fehler kommen, da diese aber jedoch immer nur schwach implementiert werden müssen, kann man dort die Anzahl entsprechend regulieren um das gewünschte Ergebnis zu erhalten.

\Rightarrow es gibt passende P'_1 und P'_2 mit $P'_1 \in \text{as-impl}(P_1) \wedge P'_2 \in \text{as-impl}(P_2) \wedge P' = P'_1 \parallel P'_2$.

□

3 Verfeinerungen für Kommunikationsfehler-Freiheit

Dieses Kapitel versucht die Präkongruenz für Error bei EIOs aus [Sch16] auf die hier betrachten MEIOs zu erweitern.

Definition 3.1 (Fehler-freie Kommunikation). Ein Kommunikationsfehler-Zustand ist lokal erreichbar in einer as-Implementierung P' eines MEIO P , wenn ein $w \in O^*$ existiert mit $p'_0 \xRightarrow{w} p' \in E'$.

Zwei MEIOs P_1 und P_2 kommunizieren Fehler-frei, wenn alle as-Implementierungen ihrer Parallelkomposition P_{12} keine Kommunikationsfehler-Zustände lokal erreichen können.

Definition 3.2 (Kommunikationsfehler-Verfeinerungs-Basisrelation). Für zwei MEIOs P_1 und P_2 mit der gleichen Signatur wird $P_1 \sqsubseteq_E^B P_2$ geschrieben, wenn ein Kommunikationsfehler-Zustand in einer as-Implementierung von P_1 nur dann lokal erreichbar ist, wenn es auch eine as-Implementierung von P_2 gibt, in der dieser Kommunikationsfehler-Zustand auch lokal erreichbar ist. Die Basisrelation stellt eine Verfeinerung bezüglich Kommunikationsfehlern dar.

\sqsubseteq_E^C bezeichnet die vollständig abstrakte Präkongruenz von \sqsubseteq_E^B bezüglich $\cdot\|\cdot$, d.h. die größte Präkongruenz bezüglich $\cdot\|\cdot$, die in \sqsubseteq_E^B enthalten ist.

Für Implementierungen P_1 und P_2 entspricht \sqsubseteq_E^B der Relation \sqsubseteq_E^B aus [Sch16].

Wie in [Sch16] werden die Fehler hier Trace basiert betrachtet. Da jedoch die Basisrelation über as-Implementierungen spricht, sind die Trace Mengen auch nicht für die MEIOs mit may-Transitionen definiert sondern nur für die Menge der möglichen as-Implementierungen eines solchen MEIOs.

Definition 3.3 (Kommunikationsfehler-Traces). Für einen MEIO P wird definiert:

- strikte Kommunikationsfehler-Traces: $StET(P) := \left\{ w \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \xRightarrow{w} p' \in E \right\}$ TODO: erzwungen Zeilenumbruch kontrollieren
- gekürzte Kommunikationsfehler-Traces: $PrET(P) := \{\text{prune}(w) \mid w \in StET(P)\}$
- Input-kritische-Traces: $MIT(P) := \left\{ wa \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \xRightarrow{w} p' \wedge a \in I \wedge p' \not\xrightarrow{a} \right\}$ TODO: erzwungen Zeilenumbruch kontrollieren

Definition 3.4 (*Kommunikationsfehler-Semantik*). *Sie P ein MEIO.*

- *Die Menge der Kommunikationsfehler-Traces von P ist $ET(P) := \text{cont}(\text{PrET}(P)) \cup \text{cont}(\text{MIT}(P))$.*
- *Die Kommunikationsfehler-geflutete Sprache von P ist $EL(P) := L(P) \cup ET(P)$.*

Für zwei MEIOs P_1, P_2 mit der gleichen Signatur wird $P_1 \sqsubseteq_E P_2$ geschrieben, wenn $ET_1 \subseteq ET_2$ und $EL_1 \subseteq EL_2$ gilt.

Hierbei ist zu beachten, dass die Mengen $StET$, $PrET$, MIT , ET und EL nur denen aus [Sch16] entsprechen, wenn P bereits eine Implementierung ist.

Satz 3.5 (*Kommunikationsfehler-Semantik für Parallelkompositionen*). *Für zwei komponierbare MEIOs P_1, P_2 und ihre Komposition P_{12} gilt:*

1. $ET_{12} = \text{cont}(\text{prune}((ET_1 \parallel EL_2) \cup (EL_1 \parallel ET_2)))$,
2. $EL_{12} = (EL_1 \parallel EL_2) \cup ET_{12}$.

Beweis. Da die Trace-Mengen und die Sprache alle nur über möglichen Traces in mindestens einer as-Implementierung der betrachteten MEIOs spricht. Muss man zu jedem Element auch die passende as-Implementierung betrachten. Wegen Lemma 2.1 gibt es aber auch immer eine passende as-Implementierung der Parallelkomposition bzw. der einzelnen Teile der Parallelkomposition. Ansonsten verläuft der Beweis analog zu dem Beweis für EIOs aus [Sch16].

1. „ \subseteq “:

Da beide Seiten der Gleichung unter der Fortsetzung cont abgeschlossen sind, genügt es ein präfix-minimales Element w von ET_{12} zu betrachten.

TODO: beweisen

1. „ \supseteq “:

TODO: beweisen

2.:

Durch die Definitionen ist klar, dass $L_i \subseteq EL_i$ und $ET_i \subseteq EL_i$ gilt. Die Argumentation startet auf den rechten Seite der Gleichung:

$$\begin{aligned}
 (EL_1 \parallel EL_2) \cup ET_{12} &\stackrel{3.4}{=} ((L_1 \cup ET_1) \parallel (L_2 \cup ET_2)) \cup ET_{12} \\
 &= (L_1 \parallel L_2) \cup \underbrace{(L_1 \parallel ET_2)}_{\substack{\subseteq (EL_1 \parallel ET_2) \\ \stackrel{1.}{\subseteq} ET_{12}}} \cup \underbrace{(ET_1 \parallel L_2)}_{\substack{\subseteq (ET_1 \parallel EL_2) \\ \stackrel{1.}{\subseteq} ET_{12}}} \cup \underbrace{(ET_1 \parallel ET_2)}_{\substack{\subseteq (EL_1 \parallel ET_2) \\ \stackrel{1.}{\subseteq} ET_{12}}} \cup ET_{12} \\
 &= (L_1 \parallel L_2) \cup ET_{12} \\
 &\stackrel{1.7}{=} L_{12} \cup ET_{12} \\
 &\stackrel{3.4}{=} EL_{12}.
 \end{aligned}$$

□

Literaturverzeichnis

- [BFLV16] Ferenc Bujtor, Sascha Fendrich, Gerald Lüttgen, und Walter Vogler, *Non-deterministic Modal Interfaces*, Theor. Comput. Sci. **642** (2016), 24–53.
- [BV15] Ferenc Bujtor und Walter Vogler, *Failure Semantics for Modal Transition Systems*, ACM Trans. Embedded Comput. Syst. **14** (2015), no. 4, 67:1–67:30.
- [Sch16] Ayleen Schinko, *Kommunikationsfehler, Verklemmung und Divergenz bei Interface-Automaten*, Bachelorarbeit, Universität Augsburg, 2016.