1 Definitionen

Stand: 12. 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 wird* $\Sigma_\tau := \Sigma \cup \{\tau\}$ *definiert. Die* Signatur *eines MEIOs entspricht* $\mathrm{Sig}(P) = (I, O)$*.*
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 das 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 Transitionssystems 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 wir die Notation $p \overset{\alpha}{\dashrightarrow} p'$ für $(p, \alpha, p') \in \dashrightarrow$ und $p \overset{\alpha}{\dashrightarrow}$ für $\exists p' : (p, \alpha, p') \in \dashrightarrow$ verwendet. Dies kann entsprechend auf Buchstaben-Sequenzen $w \in \Sigma_\tau^*$ erweitert werden zu $p \overset{w}{\dashrightarrow} p'$ ($p \overset{w}{\dashrightarrow}$) steht für die Existenz eines Laufes $p \overset{\alpha_1}{\dashrightarrow} p_1 \overset{\alpha_2}{\dashrightarrow} \dots p_{n-1} \overset{\alpha_n}{\dashrightarrow} p'$ ($p \overset{\alpha_1}{\dashrightarrow} p_1 \overset{\alpha_2}{\dashrightarrow} \dots p_{n-1} \overset{\alpha_n}{\dashrightarrow}$) mit $w = \alpha_1 \dots \alpha_n$.

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. $\widehat{w}$ steht für $w|_\Sigma$. Es wir $p \overset{w}{\Longrightarrow} p'$ für ein $w \in \Sigma^*$ geschrieben, falls $\exists w' \in \Sigma_\tau^* : \widehat{w'} = w \wedge p \overset{w'}{\dashrightarrow} p'$, und $p \overset{w}{\Longrightarrow}$, falls $p \overset{w}{\Longrightarrow} p'$ für ein beliebiges p' gilt. Falls $p_0 \overset{w}{\Longrightarrow} p$ gilt, dann wird w *Trace* genannt und p ist ein *erreichbarer Zustand*.
Analog zu $\dashrightarrow$ und $\Longrightarrow$ 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 $p \overset{a}{\not\longrightarrow}$ und $p \overset{a}{\not\dashrightarrow}$ für $\nexists p' : p \overset{a}{\longrightarrow} p'$ und $\nexists p' : p \overset{a}{\dashrightarrow} p'$ stehen soll. $p \overset{a}{\longrightarrow\!\!\Longrightarrow}$ wird geschrieben, wenn sowohl ein p' wie auch ein p'' existiert, so dass $p \overset{a}{\longrightarrow} p' \overset{\varepsilon}{\Longrightarrow} p''$ gilt. Diese Transition wird auch als *schwache-nachlaufende must-Transition* bezeichnet. Entsprechen steht $\overset{a}{\dashrightarrow\!\!\Longrightarrow}$ für die *schwache-nachlaufende may-Transition*.
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* $P_1 = (P_1, I_1, O_1, \longrightarrow_1, \dashrightarrow_1, p_{01}, E_1)$ *und* $P_2 = (P_2, I_2, O_2, \longrightarrow_2, \dashrightarrow_2, p_{02}, E_2)$ *sind* komponierbar*, falls* $O_1 \cap O_2 = \emptyset$. *Für solche MEIOs ist die* Parallelkomposition $P_{12} := P_1 \| P_2 = ((P_1 \times P_2), I, O, \longrightarrow_{12}, \dashrightarrow_{12}, (p_{01}, p_{02}), E)$ *definiert mit:* TODO: erzwungenen Zeilenumbrüche kontrollieren

- $I = (I_1 \cup I_2) \backslash (O_1 \cup O_2)$,
- $O = (O_1 \cup O_2)$,
- $$\begin{aligned} \longrightarrow_{12} = &\left\{((p_1,p_2),\alpha,(p_1',p_2)) \mid p_1 \overset{\alpha}{\longrightarrow}_1 p_1', \alpha \in \Sigma_\tau \backslash \mathrm{Synch}(P_1,P_2)\right\} \\ &\cup \left\{((p_1,p_2),\alpha,(p_1,p_2')) \mid p_2 \overset{\alpha}{\longrightarrow}_2 p_2', \alpha \in \Sigma_\tau \backslash \mathrm{Synch}(P_1,P_2)\right\} \\ &\cup \left\{((p_1,p_2),\alpha,(p_1',p_2')) \mid p_1 \overset{\alpha}{\longrightarrow}_1 p_1', p_2 \overset{\alpha}{\longrightarrow}_2 p_2', \alpha \in \mathrm{Synch}(P_1,P_2)\right\}, \end{aligned}$$
- $$\begin{aligned} \dashrightarrow_{12} = &\left\{((p_1,p_2),\alpha,(p_1',p_2)) \mid p_1 \overset{\alpha}{\dashrightarrow}_1 p_1', \alpha \in \Sigma_\tau \backslash \mathrm{Synch}(P_1,P_2)\right\} \\ &\cup \left\{((p_1,p_2),\alpha,(p_1,p_2')) \mid p_2 \overset{\alpha}{\dashrightarrow}_2 p_2', \alpha \in \Sigma_\tau \backslash \mathrm{Synch}(P_1,P_2)\right\} \\ &\cup \left\{((p_1,p_2),\alpha,(p_1',p_2')) \mid p_1 \overset{\alpha}{\dashrightarrow}_1 p_1', p_2 \overset{\alpha}{\dashrightarrow}_2 p_2', \alpha \in \mathrm{Synch}(P_1,P_2)\right\}, \end{aligned}$$
- $$\left.\begin{aligned} E = &(P_1 \times E_2) \cup (E_1 \times P_2) \qquad \textit{geerbte Kommunikationsfehler} \\ &\cup \left\{(p_1,p_2) \mid \exists a \in O_1 \cap I_2 : p_1 \overset{a}{\dashrightarrow} \wedge p_2 \overset{a}{\not\dashrightarrow}\right\} \\ &\cup \left\{(p_1,p_2) \mid \exists a \in I_1 \cap O_2 : p_1 \overset{a}{\not\dashrightarrow} \wedge p_2 \overset{a}{\dashrightarrow}\right\} \end{aligned}\right\} \textit{neue Kommunikationsfehler.}$$

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

Ein neuer Kommunikationsfehler entsteht, wenn eines der MEIOs die Möglichkeit für einen Output hat (may-Transition) und das andere MEIO den 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 und die Input-Transition durch eine may-Transition trotzdem erlaubt sein kann.
Wie bereits in [Sch16] kann es durch die Synchronisation von Inputs zu keinen neuen Kommunikationsfehler kommen, da dies in beiden Transitionssystemen keine lokal kontrollierte Aktion ist. Falls jedoch nur eines der Transitionssysteme 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* $p' R q'$,
2. $p \overset{\alpha}{\dashrightarrow} p'$ *impliziert* $q \overset{\alpha}{\dashrightarrow} q'$ *für ein* q' *mit* $p' R q'$,
3. $p \in E_P \Rightarrow q \in E_Q$.

Die Vereinigung $\sqsubseteq_{\text{as}}$ 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 ein MEIO Q und eine Implementierung P mit $P \sqsubseteq_{\text{as}} Q$, ist P eine as-Implementierung *von Q und es wird* 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* $p \xrightarrow{i} \overset{\varepsilon}{\Longrightarrow} p'$ *für ein* p' *mit* $p' R q'$,
2. $q \xrightarrow{\omega} q'$ *impliziert* $p \overset{\hat{\omega}}{\Longrightarrow} p'$ *für ein* p' *mit* $p' R q'$,
3. $p \overset{i}{\dashrightarrow} p'$ *impliziert* $q \overset{i}{\dashrightarrow} \overset{\varepsilon}{\Longrightarrow} q'$ *für ein* q' *mit* $p' R q'$,
4. $p \overset{\omega}{\dashrightarrow} p'$ *impliziert* $q \overset{\hat{\omega}}{\Longrightarrow} q'$ *für ein* q' *mit* $p' R q'$,
5. $p \in E_P \Rightarrow q \in E_Q$.

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

Die schwache Simulation erlaubt interne Aktionen beim MEIO, das die entsprechende Aktion matchen muss. Jedoch ist es zwingen notwendig, dass ein Input sofort aufgeführt wird und erst dann interne Aktinen möglich sind. Da ein Input die Reaktion auf eine Aktion ist, die die Umwelt auslöst und die nicht auf das Transitionssystem 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:* $w_1 \| w_2 := \{w \in (\Sigma_1 \cup \Sigma_2)^* \mid w|_{\Sigma_1} = w_1 \wedge w|_{\Sigma_2} = w_2\}$.
- *Für zwei Mengen von Wörtern bzw. Sprachen* $W_1 \subseteq \Sigma_1^*$ *und* $W_2 \subseteq \Sigma_2^*$ *ist deren Parallelkomposition definiert als:* $W_1 \| W_2 := \bigcup \{w_1 \| w_2 \mid w_1 \in W_1 \wedge w_2 \in W_2\}$.

Ebenso können die Definitionen der Funktionen prune und cont zum Abschneiden und Verlängern von Traces aus [Sch16] übernommen werden. Hierbei ist zu beachten, dass in dieser Arbeit ε das leere Wort und $\mathfrak{P}(M)$ die Potenzmenge der Menge M bezeichnet.

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

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

Definition 1.7 (*Sprache*). *Die* Sprache *eines MEIOs* P *ist* $L(P) := \left\{w \in \Sigma^* \mid p_0 \overset{w}{\Longrightarrow}\right\}$.

2 allgemeine Folgerungen

Proposition 2.1 (*Sprache und Implementierung*). *Die* (maximale) Sprache *eines MEIOs P ist* $L(P) = \left\{w \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \overset{w}{\Longrightarrow}\right\}$.

Beweis. Für ein $w \in L(P)$ gilt nach Definition 1.7 und den Definitionen der Transitions-Notation $\exists p_1, p_2, \ldots, p_{n-1}, p' \exists w' \in \Sigma_\tau^* : \hat{w'} = w \wedge w' = \alpha_1\alpha_2 \ldots \alpha_n \wedge p_0 \overset{\alpha_1}{\dashrightarrow} p_1 \overset{\alpha_2}{\dashrightarrow} \ldots p_{n-1} \overset{\alpha_n}{\dashrightarrow} p'$. Für ein w aus $\left\{w \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \overset{w}{\Longrightarrow}\right\}$ gilt, für ein $P' \in$ as-impl(P) das analoge nur mit must- anstatt may-Transitionen.
Aufgrund von Definition 1.3 2. kann jedes Element aus as-impl(P) nur die bereits in P vorhandenen may-Transitionen implementieren. Somit gibt es für jedes w, dass in der Sprache einer as-Implementierung von P enthalten ist auch ein entsprechendes $w \in L(P)$ mit einem Trace wie oben.
Da in $\left\{w \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \overset{w}{\Longrightarrow}\right\}$ alle Wörter enthalten sind, für dies es eine as-Implementierung gibt, die dieses Wort ausführen kann, werden somit auch alle möglichen as-Implementierungen betrachtet. Jede may-Transition aus P wird von mindestens einem $P' \in$ as-impl(P) als must-Transition implementiert. Deshalb sind auch alle Wörter, die in $L(P)$ enthalten sind in der Menge der Wörter alle as-Implementierungen von P enthalten.
Da für Implementierungen die must-Transitions-Relations-Menge die gleiche ist, wie die Menge der may-Transitions-Relationen könnte man auch für die as-Implementierungen die Definition 1.7 anwenden um die jeweilige Sprache zu bestimmen. Die Sprache eines MEIO entspricht dann der Vereinigung der Sprachen seiner as-Implementierungen. □

Proposition 2.2 (*Sprache der Parallelkomposition*). *Für zwei komponierbare MEIOs P_1 und P_2 gilt:* $L_{12} := L(P_{12}) = L_1 \| L_2$.

Beweis. Jedes Wort, dass in L_{12} enthalten ist, kann auf P_1 und P_2 projiziert werden und die Projektionen sind dann in L_1 und L_2 enthalten. In einer Parallelkomposition werden die Wörter der beiden MEIOs gemeinsam ausgeführt, falls es sich um synchronisierte Aktionen handelt, und verschränkt sequenziell, wenn es sich um unsynchronisierte Aktionen handelt. Somit sind alle Wörter aus $L_1 \| L_2$ auch Wörter der Parallelkomposition $L(P_{12})$. □

Lemma 2.3 (*as-Implementierungen und Parallelkomposition*).

1. $P'_1 \in \text{as-impl}(P_1) \wedge P'_2 \in \text{as-impl}(P_2) \Rightarrow (P'_1 \| P'_2) \in \text{as-impl}(P_1 \| P_2)$.

TODO: ausführlicher beweisen (MIA3 als mögliche Vorlage für Konstruktion)

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

TODO: STIMMT NICHT: Gegenbeispiel einbauen und alternative überlegen

Beweis.

1. Es gelte $i \in \{1, 2\}$. Jede Transition $(\dashrightarrow'_i = \longrightarrow'_i)$ 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 $P'_1 \| P'_2$ enthalten sind auch in $P_1 \| P_2$ als may-Transitionen vorhanden sind. Es können bei der Parallelkomposition von MEIOs, die die gleichen must- und may-Transitionsmengen haben (Implementierungen), keine may-Transitionen entstehen, zu denen es keine passende must-Transition gibt (Definition von $\longrightarrow_{12}$ und $\dashrightarrow_{12}$ in 1.2). $P'_1 \| P'_2$ ist also eine Implementierung.
   $P_1 \| P_2$ 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 $P'_1 \| P'_2$ die entsprechenden durch 1.3 1. geforderten Transitionen, um eine as-Implementierung von $P_1 \| P_2$ sein zu können.
   Die Kommunikationsfehler-Zustände, die P'_1 und P'_2 enthalten sind, müssten wegen Definition 1.3 3. auch in P_1 und P_2 enthalten sein. Somit enthält $P_1 \| P_2$ auch die entsprechend geerbten Kommunikationsfehler, die dann durch $P'_1 \| P'_2$ implementiert werden. In $P'_1 \| P'_2$ kann es nur zu neuen Kommunikationsfehlern kommen, wenn dies auch in $P_1 \| P_2$ möglich war. Also ist auch 1.3 3. für $P'_1 \| P'_2$ erfüllt.
   $\Rightarrow (P'_1 \| P'_2) \in \text{as-impl}(P_1 \| P_2)$.

2. Wie im Beweis zu 1. muss jede Transition aus P' in $P_1 \| P_2$ 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 beschrifte 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 das 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 $P' = P'_1 \| P'_2$ 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. 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 $P'_1 \| P'_2$ zu einer Transition führen, die P'

auf Basis der Definition 1.3 1. nicht ebenfalls implementieren muss ($\longrightarrow_{12}$ und $\dashrightarrow_{12}$ Definitionen in 1.2).
P' kann nur Kommunikationsfehler-Zustände implementieren, die bereits in $P_1 \| P_2$ möglich waren. Falls diese geerbt sind, sind die entsprechenden Fehler-Zustände auch schon in P_1 und P_2 möglich und können somit durch P_1' und P_2' implementiert werden. Die neuen Kommunikationsfehler, die von P' implementiert werden, können in P_1' und P_2' durch entsprechende nicht Implementierung von Input-Transitionen umgesetzt werden. Die Input-Transitionen können maximal als may-Transitionen in P_1 bzw. P_2 vorkommen, da es sonst in $P_1 \| P_2$ den neuen Kommunikationsfehler nicht geben würde.
$\Rightarrow$ es gibt passende P_1' und P_2' mit $P_1' \in \text{as-impl}(P_1) \wedge P_2' \in \text{as-impl}(P_2) \wedge P' = P_1' \| P_2'$.

□

Lemma 2.4 (*w-as-Implementierungen und Parallelkomposition*).

TODO: entsprechend anpassen wie 1.2 oder löschen

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

Beweis.

1. Es gelte $i \in \{1, 2\}$. Jede Transition ($\dashrightarrow_i' = \longrightarrow_i'$) einer sichtbaren Aktion in P_i' hat eine entsprechende schwache may-Transition in P_i, da Definition 1.4 3./4. 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 $P_1' \| P_2'$ enthalten sind auch in $P_1 \| P_2$ 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.
$P_1' \| P_2'$ ist mit der gleichen Begründung wie im Beweis von Satz 2.3 1. eine Implementierung.
$P_1 \| P_2$ 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 1.4 1./2. 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 $P_1 \| P_2$ und 1.4 1./2. geforderten Implementierungen von must-Transitionen in $P_1' \| P_2'$ basieren auf

den must-Transitionen der einzelnen MEIOs, die schwach in deren w-as-Implementierungen implementiert wurden. Die zusätzlichen τs, die dadurch entstehen, hindern $P_1' \| P_2'$ in der Parallelkomposition nicht daran die must-Transitionen auch entsprechend schwach zu implementieren.
Die Argumentation, wieso Definition 1.4 5. hier gilt, kann analog zu der Argumentation für Definition 1.3 3. aus dem Beweis zu Punkt 1. aus dem Lemma 2.3 übernommen werden. Die zusätzlichen τ Transitionen können nichts an geerbten und neuen Kommunikationsfehlern verändern, was nicht auch in der Parallelkomposition zu analogen Veränderungen führt. Die zu möglichen neuen Kommunikationsfehlern führende Inputs müssen entweder sofort implementiert werden oder gar nicht.
$\Rightarrow (P_1' \| P_2') \in \text{w-as-impl}(P_1 \| P_2)$.

2. Wie im Beweis zu 1. muss jede sichtbare Transition aus P' in $P_1 \| P_2$ als schwache may-Transition auftauchen (Definition 1.4 3./4.). Bei der Projektion auf die einzelnen Transitionssysteme muss die schwache Transition im jeweiligen Automaten erhalten bleiben. Wobei sich jedoch die Anzahl der τs 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 $P' = P_1' \| P_2'$ gilt.
Alle Transitionen, die in P' aufgrund von Definition 1.4 1./2. 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 (1.4) erfüllen und somit die must-Transitionen aus P_1 bzw. P_2 schwach implementieren. Hierbei muss auf die korrekte Anzahl an τs 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 $P_1' \| P_2'$ zu einer mit einer sichtbaren Aktion beschriften Transition führen, die P' auf Basis der Definition 1.4 1./2. 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.
Mit der Argumentation aus Punkte 1. dieses Beweises und den Feststellungen aus dem Beweis von Punkte 2. des Lemmas 2.3 bezüglich der Erfüllung von Definition 1.3 3., kann hier die Erfüllung von Definition 1.4 5. begründet werden.
$\Rightarrow$ es gibt passende P_1' und P_2' mit $P_1' \in \text{w-as-impl}(P_1) \wedge P_2' \in \text{w-as-impl}(P_2) \wedge P' = P_1' \| P_2'$.

□

Ein neuer Kommunikationsfehler in einer Parallelkomposition muss in einer Implementierung (as oder w-as) nicht auftauchen, auch nicht in der Parallelkomposition von Implementierungen der einzelnen Komponenten. Dies liegt daran, dass für den Input nur

gesagt wird, dass keine must-Transition für die Synchronisation der Aktion vorhanden ist. Es kann trotzdem eine may-Transition für den Input geben, die auch implementiert werden kann. Falls es aber in der Parallelkomposition zweier MEIO zu einem neuen Kommunikationsfehler kommt, dann gibt es auch immer mindestens eine Implementierung, die diesen Kommunikationsfehler enthält und es gibt auch immer mindestens ein Implementierungs-Paar der Komponenten, in deren Parallelkomposition sich dieser Kommunikationsfehler ebenfalls zeigt.

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 $p'_0 \stackrel{w}{\Longrightarrow} p' \in E'$.*
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-Basirelation*). *Für zwei MEIOs P_1 und P_2 mit der gleichen Signatur wird $P_1 \sqsubseteq^{\mathrm{B}}_E P_2$ 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.*
$\sqsubseteq^{\mathrm{C}}_E$ bezeichnet die vollständig abstrakte Präkongruenz *von $\sqsubseteq^{\mathrm{B}}_E$ bezüglich $\cdot\|\cdot$, d.h. die gröbste Präkongruenz bezüglich $\cdot\|\cdot$, die in $\sqsubseteq^{\mathrm{B}}_E$ enthalten ist.*

Für as-Implementierungen P_1 und P_2 entspricht $\sqsubseteq^{\mathrm{B}}_E$ der Relation $\sqsubseteq^{\mathrm{B}}_E$ 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 ein MEIO P wird definiert:*

- strikte Kommunikationsfehler-Traces*: $StET(P) := \left\{ w \in \Sigma^* \mid p_0 \stackrel{w}{=\!\Rightarrow} p \in E \right\}$,*
- gekürzte Kommunikationsfehler-Traces*: $PrET(P) := \{\mathrm{prune}(w) \mid w \in StET(P)\}$,*
- Input-kritische-Traces*: $MIT(P) := \left\{ wa \in \Sigma^* \mid p_0 \stackrel{w}{=\!\Rightarrow} p \wedge a \in I \wedge p \stackrel{a}{\not\rightarrow} \right\}$.*

Proposition 3.4 (*Kommunikationsfehler-Traces und Implementierung*). *Für ein MEIO P gilt:*

1. strikte Kommunikationsfehler-Traces*: $StET(P) = \left\{ w \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \stackrel{w}{\Longrightarrow} p' \in E \right\}$* TODO: erzwungen Zeilenumbruch kontrollieren

2. Input-kritische-Traces: $MIT(P) = \left\{ wa \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \stackrel{w}{\Longrightarrow} p' \wedge a \in I \wedge p' \stackrel{a}{\not\rightarrow} \right\}$ *TODO: erzwungen Zeilenumbruch kontrollieren*

Beweis.

1. Wie schon in Beweis zu 2.1 festgestellt, sind alle Abläufe, die in P via may-Transitionen möglich ist in mindestens einer as-Implementierung von P via must-Transitionen möglich. Umgekehrt ist auch jeder Ablauf, der in einer as-Implementierung von P möglich ist auch in P durch may-Transitionen möglich.
   Aufgrund des 3. Punktes der Definition 1.3 kann jede as-Implementierung von P nur Kommunikationsfehler-Zustände enthalten, die auch P enthält. Da alle möglichen Implementierungen von P in $\left\{ w \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \stackrel{w}{\Longrightarrow} p' \in E \right\}$ betrachtet werden, ist auch jeder in P durch may-Transitionen erreichbare Kommunikationsfehler-Zustand auch in mindestens einer as-Implementierung ebenfalls erreichbar, jedoch durch must-Transitionen.

2. Für jedes w in $L(P)$ gibt es mindestens eine as-Implementierung von P, die dieses w auch ausführen kann und umgekehrt (Beweis von 2.1). Falls in $MIT(P)$ mindestens ein Element gibt, gibt es in P einen Trace von w, nach dem ein Input a nicht zwingendermaßen folgen muss. Die a Transition also entweder eine may-Transition ist oder gar nicht existiert in P. Es muss also auch einen w Trace in einer as-Implementierung geben, die in einem Zustand endet, der mit dem Zustand aus P in Relation steht, in dem das a nicht erzwungen wird. Falls die Transition in P nicht vorhanden ist, muss jede as-Implementierung, die so einen w Trace enthält auch wa als Input-kritischen-Trace haben. Andernfalls handelt es sich bei a in P um eine may-Transition, die von mindestens einer as-Implementierung, die den w Trace enthält, nicht implementiert wird und somit wa auch als Input-kritischen-Trace enthält.
   Für ein $wa \in \left\{ wa \in \Sigma^* \mid \exists P' \in \text{as-impl}(P) : p'_0 \stackrel{w}{\Longrightarrow} p' \wedge a \in I \wedge p' \stackrel{a}{\not\rightarrow} \right\}$ muss es eine as-Implementierung von P geben, die diesen Input-kritischen-Trace implementiert. P muss also auch das w ausführen können zu einem Zustand, in dem P a nicht als must-Transition enthalten. Falls P a nach w nur als must-Transition enthalten würde, würde 1.3 1. die Implementierung von a erzwingen würde und somit könnte für keine as-Implementierung von P wa ein Input-kritischer-Trace sein. Es gilt also auch $wa \in MIT(P)$.

□

Definition 3.5 (*Kommunikationsfehler-Semantik*). *Sei P ein MEIO.*

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

Satz 3.6 (*Kommunikationsfehler-Semantik für Parallelkompositionen*). *Für zwei komponierbare MEIOs P_1, P_2 und ihre Komposition P_{12} gilt:*

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

Beweis. TODO: neuer Beweis auf Basis der Sprach/Trace-Definitionen ohne Implementierungen

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 2.3 gibt es aber auch immer eine passende as-Implementierung der Parallelkomposition bzw. der einzelnen Teile der Parallelkomposition.

TODO: 3. Punkt der Simulations-Definition beachten

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. Diese Element ist aufgrund der Definition der Menge der Errortraces in MIT_{12} oder in $PrET_{12}$ enthalten.

- Fall 1 ($w \in MIT_{12}$): Aus der Definition von MIT folgt, dass es eine as-Implementierung von P_{12} gibt, in der eine Aufteilung $w = xa$ existiert mit $(p_{01}, p_{02}) \stackrel{x}{\Longrightarrow} (p_1, p_2) \wedge a \in I_{12} \wedge (p_1, p_2) \stackrel{a}{\not\longrightarrow}$. Da $I_{12} = (I_1 \cup I_2) \backslash (O_1 \cup O_2)$ ist, folgt $a \in (I_1 \cup I_2)$ und $a \notin (O_1 \cup O_2)$. Es wird unterschieden, ob $a \in (I_1 \cap I_2)$ oder $a \in (I_1 \cup I_2) \backslash (I_1 \cap I_2)$ ist.

  - Fall 1a) ($a \in (I_1 \cap I_2)$): Der Ablauf der as-Implementierung der Komposition kann und auf as-Implementierung der einzelnen Transitionssysteme projiziert werden (Lemma 2.3 2.). Man erhält dann oBdA $p_{01} \stackrel{x_1}{\Longrightarrow} p_1 \stackrel{a}{\not\longrightarrow}$ und $p_{02} \stackrel{x_2}{\Longrightarrow} p_2 \stackrel{a}{\not\longrightarrow}$ oder $p_{02} \stackrel{x_2}{\Longrightarrow} p_2 \stackrel{a}{\longrightarrow}$ mit $x \in x_1 \| x_2$. Daraus kann $x_1 a \in \text{cont}(MIT_1) \subseteq ET_1$ und $x_2 a \in EL_2$ ($x_2 a \in MIT_2$ oder $x_2 a \in L_2$) gefolgert werden. Damit folgt $w \in (x_1 \| x_2) \cdot \{a\} \subseteq (x_1 a) \| (x_2 a) \subseteq ET_1 \| EL_2$, und somit ist w in der rechten Seite der Gleichung enthalten.

  - Fall 1b) ($a \in (I_1 \cup I_2) \backslash (I_1 \cap I_2)$): OBdA gilt $a \in I_1$. In der Projektion auf as-Implementierungen erhält man: $p_{01} \stackrel{x_1}{\Longrightarrow} p_1 \stackrel{a}{\not\longrightarrow}$ und $p_{02} \stackrel{x_2}{\Longrightarrow} p_2$ mit $x \in x_1 \| x_2$. Daraus folgt $x_1 a \in \text{cont}(MIT_1) \subseteq ET_1$ und $x_2 \in L_2 \subseteq EL_2$. Somit gilt $w \in (x_1 \| x_2) \cdot \{a\} \subseteq (x_1 a) \| x_2 \subseteq ET_1 \| EL_2$. Dies ist eine Teilmenge der rechten Seite der Gleichung.

- TODO: in diesem Fall as-Implementierung/Implementierung prüfen

  Fall 2 ($w \in PrET_{12}$): Aus der Definition von $PrET$ und prune folgt, dass es eine as-Implementierung und ein $v \in O_{12}^*$ gilt, so dass $(p_{01}, p_{02}) \Longrightarrow w(p_1, p_2) \overset{v}{\Longrightarrow} (p_1', p_2')$ gilt mit $(p_1', p_2') \in E_{12}$ und $w = \text{prune}(wv)$. Durch Projektion auf entsprechende as-Implementierung der Komponenten erhält man $p_{01} \overset{w_1}{\Longrightarrow} p_1 \overset{v_1}{\Longrightarrow} p_1'$ und $p_{02} \overset{w_2}{\Longrightarrow} p_2 \overset{v_2}{\Longrightarrow} p_2'$ mit $w \in w_1 \| w_2$ und $v \in v_1 \| v_2$. Aus $(p_1', p_2') \in ET_{12}$ folgt, dass es sich entweder um einen geerbten oder einen neuen Kommunikationsfehler handelt. Bei einem geerbten wäre bereits einer der beiden Zustände p_1' bzw. p_2' ein Kommunikationsfehler-Zustand gewesen. Ein neuer Kommunikationsfehler hingegen wäre durch das fehlen der Synchronisations-Erzwingung (fehlende must-Transition) in der Spezifikation einer der Komponenten entstanden. Die Spezifikation würde es also zulassen, dass die Möglichkeit in einer ihrer Implementierungen fehlt, eine synchronisierte Aktion auszuführen.

  - Fall 2a) (geerbter Kommunikationsfehler): OBdA gilt $p_1' \in E_1$. Daraus folgt, $w_1 v_1 \in StET_1 \subseteq \text{cont}(PrET_1) \subseteq ET_1$. Da $p_{02} \Longrightarrow w_2 v_2$ gilt, erhält man $w_2 v_2 \in L_2 \subseteq EL_2$. Dadurch ergibt sich $wv \in ET_1 \| EL_2$ mit $w = \text{prune}(wv)$ und somit ist w in der rechten Seite der Gleichung enthalten.

  - Fall 2b) (neuer Kommunikationsfehler): OBdA gilt $a \in I_1 \cap O_2$ mit $p_1' \overset{a}{\not\longrightarrow} \wedge p_2' \overset{a}{\longrightarrow}$ für zwei as-Implementierungen von P_1 und P_2, die in P_{12} komponiert wurden. Hierbei muss es sich nicht mehr um die passenden as-Implementierungen zu der betrachteten as-Implementierung der Komposition handeln. Da es möglich ist, dass P_1 die a-Transition als may-Transition enthält und die passende Implementierung von P_1 diese implementiert und es somit in der Parallelkomposition der Implementierungen gar nicht zu dem neuen Kommunikationsfehler kommen muss. Damit es aber in der Parallelkomposition der Spezifikationen zu den neuen Kommunikationsfehler kommt, darf die a-Transition in P_1 keine must-Transition sein und somit muss es mindestens eine as-Implementierung von P_1 geben, die diese Transition nicht implementiert.
    Es folgt also $w_1 v_1 a \in MIT_1 \subseteq ET_1$ und $w_2 v_2 a \in L_2 \subseteq EL_2$. Damit ergibt sich $wva \in ET_1 \| EL_2$, da $a \in O_1 \subseteq O_{12}$ gilt $w = \text{prune}(wva)$ und somit ist w in der rechten Seite der Gleichung enthalten.

1. „$\supseteq$“:
Wegen der Abgeschlossenheit beider Seiten der Gleichung gegenüber cont wird auch in diesem Fall nur ein präfix-minimales Element $x \in \text{prune}((ET_1 \| EL_2) \cup (EL_1 \| ET_2))$ betrachtet. Da x durch die Anwendung der prune-Funktion entstanden ist, existiert ein $y \in O_{12}^*$ mit $xy \in (ET_1 \| EL_2) \cup (EL_1 \| ET_2)$. OBdA wird davon ausgegangen, dass $xy \in ET_1 \| EL_2$ gilt, d.h. es gibt $w_1 \in ET_1$ und $w_2 \in EL_2$ mit $xy \in w_1 \| w_2$.
Im Folgenden wird für alle Fälle von xy gezeigt, dass es ein $v \in PrET_{12} \cup MIT_{12}$ gibt, das ein Präfix von xy ist und v entweder auf einen Input I_{12} endet oder $v = \varepsilon$. Damit muss v ein Präfix von x sein. ε ist Präfix von jedem Wort und sobald v mindestens einen Buchstaben enthält, muss das Ende von v vor dem Anfang von $y \in O_{12}^*$ liegen. Dadurch

ist ein Präfix von x in $PrET_{12} \cup MIT_{12}$ enthalten und somit gilt $x \in ET_{12}$, da ET die Fortsetzung der Mengenvereinigung aus $PrET$ und MIT ist.
Sei v_1 das kürzeste Präfix von w_1 in $PrET_1 \cup MIT_1$. Falls $w_2 \in L_2$, so sei $v_2 = w_2$, sonst soll v_2 das kürzeste Präfix von w_2 in $PrET_2 \cup MIT_2$ sein. Jede Aktion in v_1 und v_2 hängt mit einer aus xy zusammen. Es kann nun davon ausgegangen werden, dass entweder $v_2 = w_2 \in L_2$ gilt oder die letzte Aktion von v_1 vor oder gleichzeitig mit der letzten Aktion von v_2 statt findet. Ansonsten endet $v_2 \in PrET_2 \cup MIT_2$ vor v_1 und somit ist dieser Fall analog zu v_1 endet vor v_2.

- Fall 1 ($v_1 = \varepsilon$): Da $\varepsilon \in PrET_1 \cup MIT_1$, ist bereits in P_1 ein Kommunikationsfehler-Zustand lokal erreichbar (möglicherweise auch via may-Transitionen). $\varepsilon \in MIT_1$ ist nicht möglich, da jedes Element aus MIT nach Definition mindestens die Länge 1 haben muss. Mit der Wahl $v_2' = v' = \varepsilon$ ist v_2' ein Präfix von v_2.
- Fall 2 ($v_1 \neq \varepsilon$): Aufgrund der Definitionen von $PrET$ und MIT endet v_1 auf ein $a \in I_1$, d.h. $v_1 = v_1'a$. v' sei das Präfix von xy, das mit der letzten Aktion von v_1 endet, d.h. mit a und $v_2' = v'|_{\Sigma_2}$. Falls $v_2 = w_2 \in L_2$, dann ist v_2' ein Präfix von v_2. Falls $v_2 \in PrET_2 \cup MIT_2$ gilt, dann ist durch die Annahme, dass v_2 nicht vor v_1 endet, v_2' ein Präfix von v_2. Im Fall $v_2 \in MIT_2$ zwei man zusätzlich, dass v_2 auf $b \in I_2$ endet. Es kann jedoch $a = b$ gelten.

In allen Fällen erhält man $v_2' = v'|_{\Sigma_2}$ ist ein Präfix von v_2 und $v' \in v_1 \| v_2'$ ist ein Präfix von xy. Es kann nur für die Fälle $a \notin I_2$ gefolgert werden, dass $p_{02} \overset{v_2'}{\Longrightarrow}$ gilt.

- Fall I ($v_1 \in MIT_1$ und $v_1 \neq \varepsilon$): TODO: beweisen
- Fall II ($v_1 \in PrET_1$): 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:

$$\begin{aligned}
(EL_1 \| EL_2) \cup ET_{12} &\overset{3.5}{=} ((L_1 \cup ET_1) \parallel (L_2 \cup ET_2)) \cup ET_{12} \\
&= (L_1 \| L_2) \cup \underbrace{(L_1 \| ET_2)}_{\substack{\subseteq (EL_1 \| ET_2) \\ \overset{1.}{\subseteq} ET_{12}}} \cup \underbrace{(ET_1 \| L_2)}_{\substack{\subseteq (ET_1 \| EL_2) \\ \overset{1.}{\subseteq} ET_{12}}} \cup \underbrace{(ET_1 \| ET_2)}_{\substack{\subseteq (EL_1 \| ET_2) \\ \overset{1.}{\subseteq} ET_{12}}} \cup ET_{12} \\
&= (L_1 \| L_2) \cup ET_{12} \\
&\overset{1.7}{=} L_{12} \cup ET_{12} \\
&\overset{3.5}{=} EL_{12}.
\end{aligned}$$

□

Korollar 3.7 (*Kommunikationsfehler-Präkongruenz*). *Die Relation $\overline{\sqsubseteq}_E$ ist eine Präkongruenz bezüglich $\cdot \| \cdot$.*

Beweis. TODO: beweisen □

Literaturverzeichnis

[BFLV16] Ferenc Bujtor, Sascha Fendrich, Gerald Lüttgen, und Walter Vogler, *Nondeterministic 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.