A TRANSLATION THEOREM 1

Lemma A.1. *For all ACSP Processes M, M', CSP Processes S and actions α such that $M \xrightarrow{\alpha} M'$ and $\Gamma \vdash M \rhd S$ implies there exists a CSP Process S' and event e such that $e = m(\alpha)$ and $S \xrightarrow{e} S'$ and $\Gamma \vdash M' \rhd S'$*

Proof. Proven by rule induction on $M \xrightarrow{\alpha} M'$

case $\dfrac{M \xrightarrow{e} M' \quad e \notin E}{M \underset{E}{\parallel} N \xrightarrow{e} M' \underset{E}{\parallel} N}$ *EvParL*

From tPar, we know that

$$\Gamma, L \vdash M \rhd S \tag{1}$$
$$\Gamma, L \vdash N \rhd T \tag{2}$$
$$L \cap \Gamma = \emptyset \tag{3}$$
$$(in(N) \cap out(M)) \cup (in(M) \cap out(N)) \subseteq L \tag{4}$$
$$\Gamma \vdash M \underset{E}{\parallel} N \rhd (S \underset{E, ch(L)}{\parallel} T) \setminus ch(L) \tag{5}$$

From the reduction rule we know that

$$M \xrightarrow{e} M' \tag{6}$$
$$e \notin E \tag{7}$$

By IH with Eqs. (1) *and* (6),

$$S \xrightarrow{m(e)} S' \tag{8}$$
$$\Gamma, L \vdash M' \rhd S' \tag{9}$$

From Lemma A.2,

$$in(M') = in(M) \tag{10}$$

From Eq. (6) *and Lemma A.3,*

$$out(M') \subseteq out(M) \qquad \text{from Eq. (4)} \tag{11}$$
$$(in(N) \cap out(M')) \cup (in(M') \cap out(N)) \subseteq L \tag{12}$$

By tPar and Eqs. (2), (3), (9) *and* (12),

$$\Gamma \vdash M' \underset{E}{\parallel} N \rhd (S' \underset{E, ch(L)}{\parallel} T) \setminus ch(L)$$

We also know that $ch(L) \cap \Sigma = \emptyset$ which implies that $e \notin E, ch(L)$ and for all $e \in \Sigma$, $m(e) = e$. These allow us to construct the reduction

$$(S \underset{E, ch(L)}{\parallel} T) \setminus ch(L) \xrightarrow{m(e)} (S' \underset{E, ch(L)}{\parallel} T) \setminus ch(L)$$

case $\dfrac{M \xrightarrow{e} M' \quad N \xrightarrow{e} N' \quad e \in E \quad e \neq \tau}{M \underset{E}{\parallel} N \xrightarrow{e} M' \underset{E}{\parallel} N'}$ *EvSync*

From the reduction rule we know

$$M \xrightarrow{e} M' \tag{1}$$
$$N \xrightarrow{e} N' \tag{2}$$
$$e \in E \tag{3}$$

From tPar,

$$\Gamma, L \vdash M \rhd S \tag{4}$$
$$\Gamma, L \vdash N \rhd T \tag{5}$$
$$L \cap \Gamma = \emptyset \tag{6}$$
$$(in(N) \cap out(M)) \cup (in(M) \cap out(N)) \subseteq L \tag{7}$$
$$\Gamma \vdash M \underset{E}{\parallel} N \rhd (S \underset{E, ch(L)}{\parallel} T) \setminus ch(L) \tag{8}$$

By IH with Eqs. (1) *and* (4) *and similarly Eqs.* (2) *and* (5)

$$S \xrightarrow{m(e)} S' \tag{9}$$
$$\Gamma, L \vdash M' \rhd S' \tag{10}$$
$$T \xrightarrow{m(e)} T' \tag{11}$$
$$\Gamma, L \vdash N' \rhd T' \tag{12}$$

From Eqs. (1) *and* (2) *and Lemma A.2*

$$in(N') = in(N) \tag{13}$$
$$in(M') = in(M) \tag{14}$$

By Lemma A.3 with Eqs. (1) *and* (2)

$$out(M') \subseteq out(M) \tag{15}$$
$$out(N') \subseteq out(N) \qquad \textit{hence by transitivity with Eq.}\ (7) \tag{16}$$
$$(in(N') \cap out(M')) \cup (in(M') \cap out(N')) \subseteq L \tag{17}$$

Using tPar with Eqs. (6), (10), (12) *and* (17)

$$\Gamma \vdash M' \parallel N' \rhd (S' \underset{E, ch(L)}{\parallel} T') \setminus ch(L)$$

We know that $ch(L) \cap \Sigma = \emptyset$ *and hence* $m(e) = e \notin ch(L)$ *allowing to construct the reduction from Eqs.* (9) *and* (11)

$$(S \underset{E, ch(L)}{\parallel} T) \setminus ch(L) \xrightarrow{m(e)} (S' \underset{E, ch(L)}{\parallel} T') \setminus ch(L)$$

case $\dfrac{\textsc{EvCh}\quad j \in \mathcal{I}}{\Box_{i \in \mathcal{I}}\, e_i \to P_i \xrightarrow{e_j} P_j}$

From tChx, we know that

$$i \in \mathcal{I} \textit{ implies } \Gamma \vdash P_i \rhd S_i \tag{1}$$
$$\Gamma \vdash \Box_{i \in \mathcal{I}}\, e_i \to P_i \rhd \Box_{i \in \mathcal{I}}\, e_i \to S_i \tag{2}$$

From EvChx's premises we know there exists $j \in \mathcal{I}$ *such that* $\Box_{i \in \mathcal{I}}\, e_i \to P_i \xrightarrow{e_j} P_j$ *which by Eq.* (1) *implies* $\Gamma \vdash P_j \rhd S_j$ *allowing us to also construct the reduction*

$$(\Box_{i \in \mathcal{I}}\, e_i \to S_i) \xrightarrow{e_j} S_j$$

case $\dfrac{\textsc{IfTrue}\quad e_1 \le e_2}{\textit{if } e_1 \le e_2 \textit{ then } P \textit{ else } Q \xrightarrow{\tau} P}$

From tIf, we know that

$$\Gamma \vdash \textit{if } e_1 \le e_2 \textit{ then } P \textit{ else } Q \rhd \textit{if } e_1 \le e_2 \textit{ then } S \textit{ else } T \tag{1}$$
$$\Gamma \vdash P \rhd S \tag{2}$$
$$\Gamma \vdash Q \rhd T \tag{3}$$

Result follows by $\textit{if } e_1 \le e_2 \textit{ then } S \textit{ else } T \xrightarrow{\tau} S$ *and Eq.* (2).

AdRcv

case $\dfrac{}{l\langle Q\rangle \xrightarrow{l?P} l\langle P\rangle}$

From tLoc,

$$l \in \Gamma \tag{1}$$
$$\Gamma \vdash Q \rhd S \tag{2}$$
$$\Gamma \vdash l\langle Q\rangle \rhd S \vartriangle rec_\Gamma(l) \tag{3}$$
$$rec_\Gamma(l) = \Box_{e \in ch(l)}\, e \to (T_e \vartriangle rec_\Gamma(l)) \tag{4}$$
$$\Gamma \vdash p(e) \rhd T_e \quad \text{for all } e \in ch(l) \tag{5}$$

We know there $e' \in ch(l)$ *such that* $m(l?P) = e'$ *so* $S \vartriangle (\Box_{e \in ch(l)}\, e \to (T_e \vartriangle rec_\Gamma(l))) \xrightarrow{e'} (T_{e'} \vartriangle rec_\Gamma(l))$. *From Eq.* (1), $\Gamma \vdash P \rhd T_{e'}$, *Eq.* (5)

$$\Gamma \vdash l\langle Q\rangle \rhd T_{e'} \vartriangle rec_\Gamma(l)$$

EvLoc

case $\dfrac{P \xrightarrow{e} P'}{l\langle P\rangle \xrightarrow{e} l\langle P'\rangle}$

From tLoc

$$l \in \Gamma \tag{1}$$
$$\Gamma \vdash P \rhd S \tag{2}$$
$$\Gamma \vdash l\langle P\rangle \rhd S \vartriangle rec_\Gamma(l) \tag{3}$$
$$rec_\Gamma(l) = \Box_{e \in ch(l)}\, e \to (T_e \vartriangle rec_\Gamma(l)) \tag{4}$$
$$\Gamma \vdash p(e) \rhd T_e \quad \text{for all } e \in ch(l) \tag{5}$$

By IH with Eq. (2), $P \xrightarrow{e} P'$,

$$S \xrightarrow{m(e)} S' \tag{6}$$
$$\Gamma \vdash P' \rhd S' \tag{7}$$

By tLoc, with Eqs. (1) *and* (7)

$$\Gamma \vdash l\langle P'\rangle \rhd S' \vartriangle rec_\Gamma(l) \tag{8}$$

Rec

case $\dfrac{\sigma = [\vec{e}, (recX(\vec{y} := \vec{e}).P)/\vec{y}, X]}{recX(\vec{y} := \vec{e}).P \xrightarrow{\tau} P\sigma}$

AdSnd

case $\dfrac{}{l!P.Q \xrightarrow{l!P} Q}$

From tSnd, we know that

$$\Gamma \vdash l!P.Q \rhd e \to S \tag{1}$$
$$m(l!P) = e \tag{2}$$
$$\Gamma \vdash Q \rhd S \tag{3}$$

Result follows from Eq. (3) *and* $e \to S \xrightarrow{e} S$ *as required.*

AdParL

case $\dfrac{M \xrightarrow{h} M'}{M \underset{E}{\parallel} N \xrightarrow{h} M' \underset{E}{\parallel} N}$

From tPar, we know that

$$\Gamma, L \vdash M \rhd S \tag{1}$$
$$\Gamma, L \vdash N \rhd T \tag{2}$$
$$L \cap \Gamma = \emptyset \tag{3}$$
$$(in(N) \cap out(M)) \cup (in(M) \cap out(N)) \subseteq L \tag{4}$$
$$\Gamma \vdash M \underset{E}{\parallel} N \rhd (S \underset{E, ch(L)}{\parallel} T) \setminus ch(L) \tag{5}$$

From reduction rule, we know that

$$M \xrightarrow{h} M' \tag{6}$$

By IH with Eqs. (1) *and* (6),

$$S \xrightarrow{m(h)} S' \tag{7}$$
$$\Gamma, L \vdash M' \rhd S' \tag{8}$$

From Lemma A.2 and Eq. (6), *we know that*

$$in(M) = in(M') \tag{9}$$

From Lemma A.3 and Eq. (4)

$$(in(N) \cap out(M')) \cup (in(M) \cap out(N')) \subseteq L \tag{10}$$

Thus we can use tPar with Eqs. (2), (3), (8) *and* (10)

$$\Gamma \vdash M' \underset{E}{\parallel} N \rhd (S' \underset{E, ch(L)}{\parallel} T) \setminus ch(L) \tag{11}$$

Here there are two sub-cases

- $m(h) \notin ch(L)$ *and since* $\Sigma \cap ch(L) = \emptyset$, *we know that* $m(h) \notin E, ch(L)$. *This allows us to construct the reduction*

$$(S \underset{E, ch(L)}{\parallel} T) \setminus ch(L) \xrightarrow{m(h)} (S' \underset{E, ch(L)}{\parallel} T) \setminus ch(L)$$

- $m(h) \in ch(L)$

case $\dfrac{M \xrightarrow{l!R} M' \quad N \xrightarrow{l?R} N'}{M \underset{E}{\parallel} N \xrightarrow{\tau} M' \underset{E}{\parallel} N'}$ AdSyncL

From tPar, we know that

$$\Gamma, L \vdash M \rhd S \tag{1}$$
$$\Gamma, L \vdash N \rhd T \tag{2}$$
$$\Gamma \cap L = \emptyset \tag{3}$$
$$(in(N) \cap out(M)) \cup (in(M) \cap out(N)) \subseteq L \tag{4}$$
$$\Gamma \vdash M \underset{E}{\parallel} N \rhd (S \underset{E, ch(L)}{\parallel} T) \setminus ch(L) \tag{5}$$

From reduction rule, we know that

$$M \xrightarrow{l!R} M' \tag{6}$$
$$N \xrightarrow{l?R} N' \tag{7}$$

By IH with Eqs. (1) *and* (6) *and similarly Eqs.* (2) *and* (7)

$$S \xrightarrow{m(l?R)} S' \tag{8}$$
$$T \xrightarrow{m(l?R)} T' \tag{9}$$
$$\Gamma, L \vdash M' \rhd S' \tag{10}$$
$$\Gamma, L \vdash N' \rhd T' \tag{11}$$

We can deduce that $m(l!R) = m(l?R) = e$. *From Lemmas A.4 and A.5 with Eqs.* (6) *and* (7) *respectively,*

$$l \in in(N) \text{ and } l \in out(M) \qquad \text{hence by Eq. (4)} \tag{12}$$

$$l \in L \qquad \text{from the definition of the function } ch \tag{13}$$

$$e \in ch(L) \tag{14}$$

This implies from Eqs. (8) *and* (9)

$$(S \underset{E, ch(L)}{\parallel} T) \setminus ch(L) \xrightarrow{\tau} (S' \underset{E, ch(L)}{\parallel} T') \setminus ch(L)$$

From Lemma A.2 and Eqs. (6) *and* (7), $in(M, N) = in(M', N')$ *and*

$$(in(N') \cap out(M)) \cup (in(M') \cap out(N)) \subseteq L \tag{15}$$

By Lemma A.3 and Eqs. (6) *and* (7), $out(M, N) = out(M', N')$ *and*

$$(in(N') \cap out(M')) \cup (in(M') \cap out(N')) \subseteq L \tag{16}$$

By tPar with Eqs. (3), (10), (11) *and* (16)

$$\Gamma \vdash M' \underset{E}{\parallel} N' \rhd (S' \underset{E, ch(L)}{\parallel} T') \setminus ch(L)$$

case $\dfrac{P \xrightarrow{e}_{\textsc{EvEsc}} P' \quad c \neq e}{(\nu c)P \xrightarrow{e} (\nu c)P'}$ *From tScp, we know that*

$$\Gamma \vdash P \rhd S \tag{1}$$

From reduction rule, we know that

$$P \xrightarrow{e} P' \tag{2}$$

$$c \neq e \tag{3}$$

By IH with Eqs. (1) *and* (2)

$$S \xrightarrow{m(e)} S' \tag{4}$$

$$\Gamma \vdash P' \rhd S' \tag{5}$$

There are two sub-cases

- $c \in \mathtt{Locs}$ *but we know that* $\Sigma \cap \mathtt{Locs} = \emptyset$ *and hence* $e = m(e) \notin ch(c)$ *and we can construct the reduction* $S \setminus ch(c) \xrightarrow{m(e)} S' \setminus ch(c)$ *and from TScp2 and Eq.* (5) *it follows* $\Gamma \vdash (\nu\, c)P' \rhd S' \setminus ch(c)$
- $c \in \Sigma$, *we know from Eq.* (3) *and hence* $e = m(e) \notin \{e\}$ *and we can construct the reduction* $S \setminus \{e\} \xrightarrow{m(e)} S' \setminus \{e\}$ *and from TScp and Eq.* (5) *it follows* $\Gamma \vdash (\nu\, c)P' \rhd S' \setminus \{e\}$

case $\dfrac{P \xrightarrow{h}_{\textsc{AdEsc}} P' \quad c \notin h}{(\nu c)P \xrightarrow{h} (\nu c)P'}$ *From tScp, we know that*

$$\Gamma \vdash P \rhd S \tag{1}$$

From reduction rule, we know that

$$P \xrightarrow{h} P' \tag{2}$$

$$c \notin h \tag{3}$$

By IH with Eqs. (1) *and* (2)

$$S \xrightarrow{m(h)} S' \tag{4}$$

$$\Gamma \vdash P' \rhd S' \tag{5}$$

There are two sub-cases

- $c \in \mathtt{Locs}$ *but from Eq.* (3) *and Proposition A.6, we know that* $m(h) \notin ch(c)$ *and we can construct the reduction* $S \setminus ch(c) \xrightarrow{m(h)} S' \setminus ch(c)$ *and from TScp2 and Eq.* (5) *it follows* $\Gamma \vdash (\nu\, c)P' \rhd S' \setminus ch(c)$

- $c \in \Sigma$, *we know that* $\mathtt{Locs} \cap \Sigma = \emptyset$ *and hence* $ch(c) \neq m(h)$ *and we can construct the reduction* $S \setminus \{e\} \xrightarrow{m(h)} S' \setminus \{e\}$ *and from TScp and Eq.* (5) *it follows* $\Gamma \vdash (\nu\, c)P' \rhd S' \setminus \{e\}$

case $\dfrac{P \xrightarrow{e} P'}{(\nu e)P \xrightarrow{\tau} (\nu e)P'}$ EvHide

From tScp, we know that

$$\Gamma \vdash P \rhd S \tag{1}$$
$$\Gamma \vdash (\nu\, e)P \rhd S \setminus \{e\} \tag{2}$$

From reduction rule, we know that

$$P \xrightarrow{e} P' \tag{3}$$

By IH with Eqs. (1) *and* (3)

$$S \xrightarrow{m(e)} S' \tag{4}$$
$$m(e) = e \tag{5}$$
$$\Gamma \vdash P' \rhd S' \tag{6}$$

from Eq. (4) $S \setminus \{e\} \xrightarrow{\tau} S' \setminus \{e\}$ *and* $\Gamma \vdash (\nu\, e)P' \rhd S' \setminus \{e\}$ *follows from Eq.* (6) *and by* **??** *with Eq.* (6), *we deduce* $\Gamma \vdash (\nu\, e)P' \rhd S' \setminus \{e\}$.

case $\dfrac{P \xrightarrow{h} P'}{l\langle P\rangle \xrightarrow{h} l\langle P'\rangle}$ AdLoc

From tLoc

$$l \in \Gamma \tag{1}$$
$$\Gamma \vdash P \rhd S \tag{2}$$
$$\Gamma \vdash l\langle P\rangle \rhd S \vartriangle rec_\Gamma(l) \tag{3}$$
$$rec_\Gamma(l) = \Box_{e \in ch(l)}\, e \rightarrow (T_e \vartriangle rec_\Gamma(l)) \tag{4}$$
$$\Gamma \vdash p(e) \rhd T_e \quad \text{for all } e \in ch(l) \tag{5}$$

By IH with Eq. (2), $P \xrightarrow{h} P'$,

$$S \xrightarrow{m(h)} S' \tag{6}$$
$$\Gamma \vdash P' \rhd S' \tag{7}$$

By tLoc, with Eqs. (1) *and* (7)

$$\Gamma \vdash l\langle P'\rangle \rhd S' \vartriangle rec_\Gamma(l) \tag{8}$$

We can construct from Eq. (6) *the reduction* $S \vartriangle rec_\Gamma(l) \xrightarrow{m(h)} S' \vartriangle rec_\Gamma(l)$

□

Lemma A.2. $M \xrightarrow{\alpha} M'$ *implies* $in(M) = in(M')$

Proof. Proven by rule induction on $M \xrightarrow{\alpha} M'$

case *evCh, Rec, ifTrue, adSnd Analogous as* $in(P) = \emptyset$.

case $\dfrac{}{l\langle Q\rangle \xrightarrow{l?P} l\langle P\rangle}$ AdRcv *AdLoc EvLoc*

We know that $in(l\langle P\rangle) = l$ *and* $in(l\langle Q\rangle) = l$, *as required.*

AdSyncL

case $\dfrac{M \xrightarrow{l!R} M' \quad N \xrightarrow{l?R} N'}{M \underset{E}{\parallel} N \xrightarrow{\tau} M' \underset{E}{\parallel} N'}$, evSync, evParL, adParL

By IH, we know that

$$in(M) = in(M')$$
$$in(N) = in(N')$$

We also know that

$$\begin{aligned} in(M \underset{E}{\parallel} N) &= in(M) \cup in(N) \\ &= in(M') \cup in(N') \\ &= in(M' \underset{E}{\parallel} N') \end{aligned}$$

EvHide

case $\dfrac{P \xrightarrow{e} P'}{(ve)P \xrightarrow{\tau} (ve)P'}$, adEsc evEsc

By IH, we know that

$$in(P) = in(P')$$
$$in(P) - \{e\} = in(P') - \{e\}$$

□

Lemma A.3. *$M \xrightarrow{\alpha} M'$ then $out(M') \subseteq out(M)$*

Proof. Proven by rule induction on $M \xrightarrow{e} M'$

IfTrue

case $\dfrac{e_1 \leq e_2}{if\, e_1 \leq e_2 \text{ then } P \text{ else } Q \xrightarrow{\tau} P}$ pChx Rec

By definition $out(if\, e_1 \leq e_2 \text{ then } P \text{ else } Q) = out(P) \cup out(Q)$ and hence $out(P) \subseteq out(P) \cup out(Q)$

AdSyncL

case $\dfrac{M \xrightarrow{l!R} M' \quad N \xrightarrow{l?R} N'}{M \underset{E}{\parallel} N \xrightarrow{\tau} M' \underset{E}{\parallel} N'}$, evSync, evParL, adParL

By IH, we know that

$$out(M') \subseteq out(M) \quad \text{and } out(N') \subseteq out(N)$$
$$out(M') \cup out(N') \subseteq out(M) \cup out(N)$$
$$out(M' \underset{E}{\parallel} N') \subseteq out(M \underset{E}{\parallel} N)$$

AdSnd

case $\dfrac{}{l!P.Q \xrightarrow{l!P} Q}$

By definition $out(l!P.Q) = \{l\} \cup out(Q)$.

AdRcv

case $\dfrac{}{l\langle Q\rangle \xrightarrow{l?P} l\langle P\rangle}$ EvLoc AdLoc

By definition, we know that communicated processes do not communicate themselves i.e., $out(P) = \emptyset$ and thus $out(l\langle P\rangle) = out(P) = \emptyset$.

case $\dfrac{P \xrightarrow{e} P'}{(\nu e)P \xrightarrow{\tau} (\nu e)P'}$ (EvHide), adEsc evEsc

By IH we know that

$$out(P') \subseteq out(P)$$
$$out(P') - \{e\} \subseteq out(P) - \{e\}$$
$$out((\nu\, e)P') \subseteq out((\nu\, e)P)$$

□

Lemma A.4. *$M \xrightarrow{l!R} M'$ implies $l \in out(M)$*

Proof. Proven by rule induction on $M \xrightarrow{l!R} M'$

case *EvCh,EvParL,EvSync,Rec,EvHide,EvEsc,IfTrue,EvLoc,AdSyncL,AdRcv trivially true as the reduction does not include a send event.*

case $\dfrac{P \xrightarrow{h} P' \quad c \notin h}{(\nu c)P \xrightarrow{h} (\nu c)P'}$ (AdEsc)

The only applicable case is when h =l!R, which by IH we know that

$$l \in out(P)$$

From $c \notin h$ we know that $l \neq c$ and hence

$$l \in out(P) - \{c\}$$
$$l \in out((\nu\, c)P)$$

case $\dfrac{}{l!P.Q \xrightarrow{l!P} Q}$ (AdSnd)

From definition $out(l!R.Q) = \{l\} \cup out(Q)$.

case $\dfrac{M \xrightarrow{h} M'}{M \underset{E}{\parallel} N \xrightarrow{h} M' \underset{E}{\parallel} N}$ (AdParL)

By IH with $M \xrightarrow{h} M'$, it follows that

$$l \in out(M)$$
$$l \in out(M) \cup out(N)$$
$$l \in out(M \underset{E}{\parallel} N)$$

case $\dfrac{P \xrightarrow{h} P'}{l\langle P\rangle \xrightarrow{h} l\langle P'\rangle}$ (AdLoc) *By IH, we know that $l \in out(P)$, which by definition imply that $l \in out(l\langle P\rangle)$.*

□

Lemma A.5. *$M \xrightarrow{l?R} M'$ implies $l \in in(M)$*

Proof. Proven by rule induction on $M \xrightarrow{l!R} M'$

case *EvCh,EvParL,EvSync,Rec,EvHide,EvEsc,IfTrue,EvLoc,AdSyncL,AdSnd trivially true as the reduction does not include an event.*

AdEsc

case $\dfrac{P \xrightarrow{h} P' \quad c \notin h}{(\nu c)P \xrightarrow{h} (\nu c)P'}$

The only applicable case is when $h = l?R$, which by IH we know that

$$l \in in(P)$$

From $c \notin h$ we know that $l \neq c$ and hence

$$l \in in(P) - \{c\}$$
$$l \in in((\nu\, c)P)$$

AdRcv

case $\dfrac{}{l\langle Q\rangle \xrightarrow{l?P} l\langle P\rangle}$

follows directly from $in(l\langle Q\rangle) = \{l\}$.

AdvParL

case $\dfrac{M \xrightarrow{l?R} M'}{M \underset{E}{\parallel} N \xrightarrow{l?R} M' \underset{E}{\parallel} N}$ *(omitted from the language list)*

By IH with $M \xrightarrow{l?R} M'$, it follows that

$$l \in in(M)$$
$$l \in in(M) \cup in(N)$$
$$l \in in(M \underset{E}{\parallel} N)$$

AdLoc

case $\dfrac{P \xrightarrow{h} P'}{l\langle P\rangle \xrightarrow{h} l\langle P'\rangle}$ *By IH, we know that $l \in in(P)$, which by definition imply that $l \in in(l\langle P\rangle)$.*

□

Proposition A.6. *For a location c and adaptation event h,*

$$c \notin h \text{ implies } m(h) \notin ch(c)$$

B TRANSLATION THEOREM 2

Lemma B.1. *For all ACSP processes M, CSP processes S, S' and CSP events e such that $S \xrightarrow{e} S'$ and $\Gamma \vdash M \rhd S$ implies there exists α such that $m(\alpha) = e$ and $M \xrightarrow{\alpha} M'$ and $\Gamma \vdash M' \rhd S'$*

Proof. Proven by rule induction on $\Gamma \vdash M \rhd S$

case $\dfrac{i \in \mathcal{I} \text{ implies } \Gamma \vdash P_i \rhd S_i}{\Gamma \vdash \Box_{i \in \mathcal{I}}\, e_i \to P_i \rhd \Box_{i \in \mathcal{I}}\, e_i \to S_i}$ *tChx*

From tChx we know

$$M = \Box_{i \in \mathcal{I}}\, e_i \to P_i \tag{1}$$

$$S = \Box_{i \in \mathcal{I}}\, e_i \to S_i \tag{2}$$

$$i \in \mathcal{I} \text{ implies } \Gamma \vdash P_i \rhd S_i \tag{3}$$

From the structures of both S and M, we know that the only reduction possible is to resolve the external choice, i.e.,

$$\Box_{i \in \mathcal{I}}\, e_i \to S_i \xrightarrow{e_j} S_j \tag{4}$$

We know that for all events $e_i \in \Sigma$, $m(e_i) = e_i$, which allow us to construct the reduction

$$\Box_{i \in \mathcal{I}}\, e_i \to P_i \xrightarrow{m(e_j)} P_j \tag{5}$$

Result $\Gamma \vdash P_j \rhd S_j$ follows from Eq. (3).

case $\dfrac{\Gamma \vdash M \rhd S}{\Gamma \vdash (\nu e)M \rhd S \setminus \{e\}}$ *tScp*

From tScp, we know

$$\Gamma \vdash M \rhd S \tag{1}$$

From operational semantics of CSP, this means that $S \xrightarrow{e'} S'$. By IH

$$m(\alpha) = e' \tag{2}$$

$$M \xrightarrow{\alpha} M' \tag{3}$$

$$\Gamma \vdash M' \rhd S' \tag{4}$$

By tScp and Eq. (3)

$$\Gamma \vdash (\nu\, e)M' \rhd S' \setminus \{e\} \tag{5}$$

For the reduction, there are two sub-cases

- *$e = e'$ and hence $S \setminus \{e\} \xrightarrow{\tau} S' \setminus \{e\}$. This means that $\alpha = e' = e$ and hence by evHide and Eq.* (3)*, we can construct the reduction*

$$(\nu\, e)M \xrightarrow{\tau} (\nu\, e)M' \tag{6}$$

- *$e \neq e'$ and hence $S \setminus \{e\} \xrightarrow{e'} S' \setminus \{e\}$ by evEsc or AdEsc (depending on the structure of e') and Eq.* (3)*, we can construct the reduction*

$$(\nu\, e)M \xrightarrow{\alpha} (\nu\, e)M' \tag{7}$$

case $\dfrac{\Gamma \vdash P \rhd S}{\Gamma \vdash recX(\vec{y} := \vec{e}).P \rhd recX(\vec{y} := \vec{e}).S}$ *tRec*

case $\dfrac{\Gamma \vdash P \rhd S \quad \Gamma \vdash Q \rhd T}{\Gamma \vdash if e_1 \leq e_2 \text{ then } P \text{ else } Q \rhd if e_1 \leq e_2 \text{ then } S \text{ else } T}$ *tIf*

From tIf, we know

$$\Gamma \vdash P \rhd S \tag{1}$$

$$\Gamma \vdash Q \rhd T \tag{2}$$

From the structure of S and M, we infer that the only possible reduction is to resolve the if statement. Without loss of generality, we assume that $e_1 \leq e_2$, then

$$if\, e_1 \leq e_2\ then\ P\ else\ Q \xrightarrow{\tau} P \tag{3}$$

$$if\, e_1 \leq e_2\ then\ S\ else\ T \xrightarrow{\tau} S \tag{4}$$

Note that $m(\tau) = \tau$ and result follows from Eq. (1).

case $\dfrac{\Gamma \vdash M \rhd S}{\Gamma \vdash (\nu l)M \rhd S \setminus ch(l)}\ \tau Scp2$

From tScp2, we know that

$$\Gamma \vdash M \rhd S \tag{1}$$

From operational semantics of CSP, this means that $S \xrightarrow{e'} S'$. By IH

$$m(\alpha) = e' \tag{2}$$

$$M \xrightarrow{\alpha} M' \tag{3}$$

$$\Gamma \vdash M' \rhd S' \tag{4}$$

By tScp2 and Eq. (3)

$$\Gamma \vdash (\nu\, l)M' \rhd S' \setminus ch(l) \tag{5}$$

For the reduction, there are two sub-cases

- *$e \in ch(l)$ and hence $S \setminus ch(l) \xrightarrow{\tau} S' \setminus ch(l)$. This means that $\alpha = e' = e$ and hence by evHide and Eq.* (3)*, we can construct the reduction*

$$(\nu\, e)M \xrightarrow{\tau} (\nu\, e)M' \tag{6}$$

- *$e' \notin ch(l)$ and hence $S \setminus ch(l) \xrightarrow{e'} S' \setminus ch(l)$ by evEsc or AdEsc (depending on the structure of e') and Eq.* (3)*, we can construct the reduction*

$$(\nu\, l)M \xrightarrow{\alpha} (\nu\, l)M' \tag{7}$$

case $\dfrac{l \in \Gamma \quad m(l!P) = e \quad \Gamma \vdash Q \rhd S}{\Gamma \vdash l!P.Q \rhd e \rightarrow S}\ \tau Snd$

From tSnd, we know that

$$l \in \Gamma \tag{1}$$

$$m(l!P) = e \tag{2}$$

$$\Gamma \vdash Q \rhd S \tag{3}$$

From the structure of both M and S we know that the only reduction possible is AdSnd and the prefix

$$l!P.Q \xrightarrow{l!P} Q \tag{4}$$

$$e \rightarrow S \xrightarrow{e} S \tag{5}$$

Result follows from Eqs. (2) *to* (4).

case $\dfrac{l \in \Gamma \quad \Gamma \vdash P \rhd S}{\Gamma \vdash l\langle P\rangle \rhd S \triangle rec_\Gamma(l)}\ \tau Loc$

From tLoc, we know that

$$l \in \Gamma \tag{1}$$

$$\Gamma \vdash P \rhd S \tag{2}$$

From the structure of S, we know there are two possible reductions

- $S \xrightarrow{e} S'$. *By IH with Eq.* (2),

$$P \xrightarrow{\alpha} P' \tag{3}$$
$$m(\alpha) = e \tag{4}$$
$$\Gamma \vdash P' \rhd S' \tag{5}$$

By tLoc with Eqs. (1) *and* (5),

$$\Gamma \vdash l\langle P' \rangle \rhd S' \triangle rec_\Gamma(l) \tag{6}$$

and by AdLoc or EvLoc, depending on the structure of e, we construct the reduction

$$l\langle P \rangle \xrightarrow{\alpha} l\langle P' \rangle \tag{7}$$

- $rec_\Gamma(l) \xrightarrow{e} S'$ *where*

$$rec_\Gamma(l) = \Box_{e \in ch(l)}\, e \to (T_e \triangle rec_\Gamma(l)) \tag{8}$$
$$e \in ch(l) \text{ implies } \Gamma \vdash p(e) \rhd T_e \tag{9}$$

Given the structure of $rec_\Gamma(l)$*, the only possible reduction is to resolve the choice, such that there is an event* $e \in ch(l)$*,*

$$S' = T_e \triangle rec_\Gamma(l) \tag{10}$$
$$\Gamma \vdash p(e) \rhd T_e \tag{11}$$

From tLoc and Eqs. (1) *and* (11)

$$\Gamma \vdash l\langle p(e) \rangle \rhd T_e \triangle rec_\Gamma(l) \tag{12}$$

Through AdRcv, we can construct the reduction

$$l\langle P \rangle \xrightarrow{l?p(e)} l\langle p(e) \rangle \tag{13}$$

Recall that $p(e) = P$ *if* $m(l?P) = e$*.*

case $\dfrac{\Gamma, L \vdash M \rhd S \quad \Gamma, L \vdash N \rhd T \quad L \cap \Gamma = \emptyset \qquad (in(N) \cap out(M)) \cup (in(M) \cap out(N)) \subseteq L}{\Gamma \vdash M \underset{\Sigma}{\|} N \rhd (S \underset{\Sigma, ch(L)}{\|} T) \setminus ch(L)}\ {}^{\tau PAR}$

From tPar, we know

$$\Gamma, L \vdash M \rhd S \tag{1}$$
$$\Gamma, L \vdash N \rhd T \tag{2}$$
$$L \cap \Gamma = \emptyset \tag{3}$$
$$(in(N) \cap out(M)) \cup (in(M) \cap out(N)) \subseteq L \tag{4}$$

From the operational semantics of CSP and the structure of S, we know that there are three sub-cases

- $S \xrightarrow{e} S'$ *and* $e \notin E, ch(L)$*. By IH with Eq.* (1)

(5)

$$M \xrightarrow{\alpha} M' \tag{6}$$
$$m(\alpha) = e \tag{7}$$
$$\Gamma, L \vdash M' \rhd S' \tag{8}$$

By AdParL or evParL depending on the structure of e, we can construct the reduction

(9)

$$M \underset{E}{\|} N \xrightarrow{\alpha} M' \underset{E}{\|} N \tag{10}$$

From Lemmas A.2 and A.3 with Eq. (6) *implies that* $out(M') \subseteq out(M)$ *and* $in(M') \subseteq in(M)$ *which means that Eq.* (4)

$$(in(N) \cap out(M')) \cup (in(M') \cap out(N)) \subseteq L \tag{11}$$

Result follows from tPar with Eqs. (2), (3), (8) *and* (11)*.*

- $T \xrightarrow{e} T'$ *and* $e \notin E, ch(L)$ *- analogous to the previous case.*

- $S \xrightarrow{e} S', T \xrightarrow{e} T'$ *and* $e \in E, ch(L)$ *By IH with Eqs.* (1) *and* (2)

$$M \xrightarrow{\alpha} M' \tag{12}$$

$$N \xrightarrow{\alpha'} N' \tag{13}$$

$$m(\alpha) = e \qquad \text{and } m(\alpha') = e \tag{14}$$

$$\Gamma, L \vdash M' \rhd S' \tag{15}$$

$$\Gamma, L \vdash N' \rhd T' \tag{16}$$

There are two sub-cases

– $e \in E$ *and* $\alpha = \alpha' = m(\alpha) = m(\alpha') = e$ *and through evSync with Eqs.* (12) *and* (13) *we can construct the reduction*

$$M \underset{E}{\parallel} N \xrightarrow{e} M' \underset{E}{\parallel} N' \tag{17}$$

– $e \in ch(L)$. *Note that this means that there exists a location* l *and process* R *such that*

$$l \in (in(N) \cap out(M)) \cup (in(M) \cap out(N)) \tag{18}$$

$$m(l!R) = m(l!R) = e \tag{19}$$

From the well-formed assumption, we know that $out(M) \cap out(N) = \emptyset$ *(similarly for in). From the contra positive of Lemmas A.4 and A.5, we can infer that either* $\alpha = l?P$ *and* $\alpha' = l!P$ *or vice-versa.*
From AdSyncL with Eqs. (12) *and* (13) *we can construct*

$$M \underset{E}{\parallel} N \xrightarrow{\tau} M' \underset{E}{\parallel} N' \tag{20}$$

From Lemmas A.2 and A.3 with Eqs. (12) *and* (13) *implies that* $out(M') \subseteq out(M)$ *and* $in(M') \subseteq in(M)$ *(similarly for* N, N'*) which means that Eq.* (4)

$$(in(N') \cap out(M')) \cup (in(M') \cap out(N')) \subseteq L \tag{21}$$

Result follows from tPar with Eqs. (3), (15), (16) *and* (21).

□

C MOTIVATING EXAMPLE ENCODING

C.1 Requirement 1

Restoration Area

$R_0 = vis_{(\mathrm{ra}, c_2)} \rightarrow R_0 \;\square\; vis_{(c_2, \mathrm{ra})} \rightarrow R_0$

$R_1 = SKIP$

$R(v) = (vis_{(\mathrm{ra}, c_2)} \rightarrow R(v-1)) \;\square\; (v > 0 \,\&\, vis_{(c_2, \mathrm{ra})} \rightarrow R(v+1))$

$\Pi_r = grd_{(s, c_2)} \rightarrow RA!R_0.\,\Pi_r \quad \square \quad grd_{(c_2, s)} \rightarrow RA!R_1.\,\Pi_r$

$$ResArea = (\nu\, RA)(\Pi_r \underset{\{vis_{(c_2,\mathrm{ra})}, vis_{(\mathrm{ra},c_2)}\}}{\|} R(0) \underset{\{vis_{(c_2,\mathrm{ra})}, vis_{(\mathrm{ra},c_2)}\}}{\|} RA\langle R_1\rangle)$$

Corridor 2

$$C(v_{c_2}, v_{\mathrm{ra}}, g_{c_2}) = \square \begin{cases} & vis_{(s,c_2)} \rightarrow C(v_{c_2}+1, v_{\mathrm{ra}}, g_{c_2}) \\ v_{c_2} > 0 \,\& & vis_{(c_2,s)} \rightarrow C(v_{c_2}-1, v_{\mathrm{ra}}, g_{c_2}) \\ v_{c_2} > 0 \,\& & vis_{(c_2,\mathrm{ra})} \rightarrow C(v_{c_2}-1, v_{\mathrm{ra}}+1, g_{c_2}) \\ v_{\mathrm{ra}} > 0 \,\& & vis_{(\mathrm{ra},c_2)} \rightarrow C(v_{c_2}+1, v_{\mathrm{ra}}-1, g_{c_2}) \\ g_{c_2} = 0 \,\& & grd_{(s,c_2)} \rightarrow C(v_{c_2}, v_{\mathrm{ra}}, g_{c_2}+1) \\ g_{c_2} > 0 \,\& & grd_{(c_2,s)} \rightarrow C(v_{c_2}, v_{\mathrm{ra}}, g_{c_2}-1) \\ & cnt.v_{c_2} + v_{\mathrm{ra}} \rightarrow C(v_{c_2}, v_{\mathrm{ra}}, g_{c_2}) \end{cases}$$

$$C2_0 = \underset{t \in \{vis, grd\}}{\square} \left(t_{(s,c_2)} \rightarrow C2_0 \;\square\; t_{(c_2,s)} \rightarrow C2_0\right) \quad \square\; vis_{(\mathrm{ra},c_2)} \rightarrow C2_0 \;\square\; vis_{(c_2,\mathrm{ra})} \rightarrow C2_0$$

$C2_1 = vis_{(s,c_2)} \rightarrow C2_1 \;\square\; vis_{(c_2,s)} \rightarrow C2_1 \;\square\; vis_{(c_2,\mathrm{ra})} \rightarrow C2_1 \;\square\; vis_{(\mathrm{ra},c_2)} \rightarrow C2_1 \;\square\; grd_{(s,c_2)} \rightarrow C2_1$

$\Pi_{c_2} = vis_{(s,c_2)} \rightarrow cnt.v \rightarrow \text{if } v == 1 \text{ then } C2!C2_1.\,\Pi_{c_2} \text{ else } \Pi_{c_2}$
$\quad \square\; vis_{(c_2,s)} \rightarrow cnt.v \rightarrow \text{if } v == 0 \text{ then } C2!C2_0.\,\Pi_{c_2} \text{ else } \Pi_{c_2}$
$\quad \square\; vis_{(c_2,\mathrm{ra})} \rightarrow \Pi_{c_2} \;\square\; vis_{(\mathrm{ra},c_2)} \rightarrow \Pi_{c_2}$
$\quad \square\; grd_{(c_2,s)} \rightarrow \Pi_{c_2} \;\square\; grd_{(s,c_2)} \rightarrow \Pi_{c_2}$

$$Corr2 = (\nu\, C2)(\nu\, cnt)(\Pi_{c_2} \underset{\Sigma, cnt}{\|} (C(0) \underset{\Sigma}{\|} C2\langle C2_0\rangle))$$

$$Requirement_1 = Corr2 \underset{\{vis_{(\mathrm{ra},c_2)}, vis_{(c_2,\mathrm{ra})}, grd_{(s,c_2)}, grd_{(c_2,s)}\}}{\|} ResArea$$

C.2 Requirement 2

Room A - D

$$R_0(rm) = \underset{t \in \{vis, emp, grd\}}{\square} \left(t_{(rm,\_\_)} \rightarrow R_0(rm)) \;\square\; (t_{(\_\_,rm)} \rightarrow R_0(rm))\right)$$

$R_1(rm) = SKIP$

$$C_{EA}(rm, n) = \underset{t \in \{vis, emp, grd\}}{\square} \left(t_{(\_\_,rm)} \rightarrow C_{EA}(rm, n+1)) \;\square\; t_{(rm,\_\_)} \rightarrow C_{EA}(rm, n-1)\right) \quad \square \quad count.n \rightarrow C_{EA}(rm, n)$$

$M(rm, n) = (max.n \rightarrow M(rm, n)) \;\square\; (inc.rm \rightarrow M(rm, n+1)) \;\square\; (dec.rm \rightarrow M(rm, n-1))$

$$\Pi_{EA}(rm) = \Box_{t \in \{vis, emp, grd\}} t_{(\_\_, rm)} \rightarrow count?n \rightarrow max?m \rightarrow (\text{if } n == m$$
$$\text{then } eaR!R_0.\, \Pi_{EA}(rm)$$
$$\text{else if } n > m \rightarrow eaR!\,(dec.adjRm \rightarrow R_1(rm))\,.\, \Pi_{EA}(rm)$$
$$\text{else } \Pi_{EA}(rm))$$
$$\Box_{t \in \{vis, emp, grd\}} t_{(rm, \_\_)} \rightarrow count?n \rightarrow max?m \rightarrow (\text{if } n == m - 1$$
$$\text{then } eaR!R_0.\, \Pi_{EA}(rm)$$
$$\text{else if } n >= m \text{ then } eaR!\,(inc.adjRm \rightarrow R_1(rm))\,.\, \Pi_{EA}(rm)$$
$$\text{else } \Pi_{EA}(rm))$$

where $adjRm = (|EA \Rightarrow EB, EB \Rightarrow ED, ED \Rightarrow EC, EC \Rightarrow EA|)$

$$Movement = \Box_{t \in \{vis, emp, grd\}} \begin{cases} (t_{(EA,EB)} \rightarrow Movement) \,\Box\, (t_{(EB,EA)} \rightarrow Movement) \\ (t_{(EB,ED)} \rightarrow Movement) \,\Box\, (t_{(ED,EB)} \rightarrow Movement) \\ (t_{(ED,EC)} \rightarrow Movement) \,\Box\, (t_{(EC,c)} \rightarrow Movement) \\ (t_{(EA,c)} \rightarrow Movement) \,\Box\, (t_{(c,EA)} \rightarrow Movement) \end{cases}$$

$Room_{EA}(rm) = \text{let}$

$$adtMgr = (\nu\, max)\left(\left((\nu\, count)\left(\Pi_{EA}(rm) \underset{count, goto}{\|} \left(C_{EA}(rm, 0) \underset{goto}{\|} Movement\right)\right)\right) \underset{max}{\|} M(rm, MaxAgents)\right)$$

within

$$eaR\langle R_0(rm)\rangle \underset{goto}{\|} adtMgr$$

$Requirement_2 = \text{let}$

$$AB = \left(Room_{EA}(EA) \underset{t_{(EA,EB)}, t_{(EB,EA)}, inc.EB, dec.EB}{\|} Room_{EA}(EB)\right)$$
$$CD = \left(Room_{EA}(EC) \underset{t_{(EC,ED)}, t_{(ED,EC)}, inc.ED, dec.ED}{\|} Room_{EA}(ED)\right)$$

within

$$(\nu\, inc, dec)\left(AB \underset{|t_{(EB,ED)}, t_{(ED,EB)}, inc, dec|}{\|} CD\right)$$

C.3 Requirement 3

HVAC

$$\Pi_H = (conn_{hvac} \rightarrow hvac!Conn.\, \Pi_H) \,\Box\, (disconn_{hvac} \rightarrow hvac!Dis.\, \Pi_H)$$

$$HVAC = (\nu\, hvac)\,(hvac\langle Conn\rangle \parallel \Pi_H)$$

Access Point

$$AP_0 = \Box_{t \in \{emp, vis\}} (conn_t \rightarrow AP_0 \,\Box\, disconn_t \rightarrow AP_0)$$

$$AP_1 = \left(conn_{emp} \rightarrow AP_1 \,\Box\, disconn_{emp} \rightarrow AP_1\right)$$

$$C_{AP}(v, e, h) = (conn_{vis} \rightarrow C_{AP}(v + 1, e, h)) \quad \Box \quad (conn_{emp} \rightarrow C_{AP}(0, e + 1, h))$$
$$\Box\, (e > 0 \;\&\; disconn_{emp} \rightarrow C_{AP}(v, e - 1)) \quad \Box \quad (v > 0 \;\&\; disconn_{vis} \rightarrow C_{AP}(v - 1, e, h))$$
$$\Box\, (conn_{hvac} \rightarrow C_{AP}(0, e, true)) \quad \Box \quad (disconn_{hvac} \rightarrow C_{AP}(v, e, false))$$
$$\Box\, (\text{if } e > 0 \text{ then } count.Secure.e.h \rightarrow C_{AP}(v, e, h) \text{ else } count.Insecure.v.h \rightarrow C_{AP}(v, e, h))$$

$$\begin{aligned}\Pi_{AP} = \quad & (conn_{emp} \rightarrow count.s?c?h \rightarrow \text{if } \neg s \text{ then } ap!disconn_{vis} \rightarrow conn_{hvac} \rightarrow AP_1.\,\Pi_{AP} \\ & \qquad \text{else if } \neg h \text{ then } ap!conn_{hvac} \rightarrow AP_1.\,\Pi_{AP} \\ & \qquad \text{else if } c == 1 \text{ and } h \text{ then } ap!AP_1.\,\Pi_{AP} \\ & \qquad \text{else } \Pi_{AP}) \\ & \Box\ (disconn_{vis} \rightarrow count?s?c?h \rightarrow \text{if } \neg h \text{ and } c == 0 \text{ then } ap!(conn_{hvac} \rightarrow AP_0).\,\Pi_{AP} \text{ else } \Pi_{AP}) \\ & \Box\ (conn_{vis} \rightarrow count?s?c?h \rightarrow \text{if } h \text{ then } ap!(disconn_{hvac} \rightarrow AP_0).\,\Pi_{AP} \text{ else } \Pi_{AP}) \\ & \Box\ (disconn_{emp} \rightarrow count?s?c?h \text{ if } s == Secure \text{ and } c == 0 \text{ then } ap!AP_0.\,\Pi_{AP} \text{ else } \Pi_{AP})\end{aligned}$$

$$\begin{aligned}AccessPt = & \text{let } adtMgr = (\nu\, count)\left(C_{AP}(0,0) \underset{\{conn_{emp}, conn_{vis}, disconn_{emp}, disconn_{vis}\}}{\|} \Pi_{AP}\right) \\ & \text{within} \\ & (\nu\, ap)\left(ap\langle conn_{hvac} \rightarrow AP_0\rangle \underset{\{connect, disconnect\}}{\|} adtMgr\right)\end{aligned}$$

Stairs

$$\begin{aligned}Stairs(v,e) = & (v > 0\ \&\ Vis_{(c_2,s)} \rightarrow disconn_{vis} \rightarrow Stairs(v-1,e)) \\ & \Box\ (v > 0\ \&\ vis_{(s,c)} \rightarrow Stairs(v,e)) \\ & \Box\ (e > 0\ \&\ emp_{(c_2,s)} \rightarrow disconn_{emp} \rightarrow Stairs(v,e-1)) \\ & \Box\ (e > 0\ \&\ emp_{(s,c)} \rightarrow Stairs(v,e-1)) \\ & \Box\ (vis_{(s,c_2)} \rightarrow conn_{vis} \rightarrow Stairs(v+1,e)) \\ & \Box\ (vis_{(c,s)} \rightarrow Stairs(v+1,e)) \\ & \Box\ (emp_{(s,c_2)} \rightarrow conn_{emp} \rightarrow Stairs(v,e+1)) \\ & \Box\ (emp_{(c,s)} \rightarrow Stairs(v,e+1))\end{aligned}$$

$$Requirement_3 = \left(HVAC \underset{\{conn_{hvac}, disconn_{hvac}\}}{\|} AccessPt\right) \underset{\{conn_{emp}, conn_{vis}, disconn_{emp}, disconn_{vis}\}}{\|} Stairs(0,0)$$

D FDR CODE FOR VERIFYING REQUIREMENT 1

```
transparent normal

datatype Countable = Vis | Emp | Grd | Secure | Insecure | HVAC

datatype Room = Corr2 | Str | Corr | RA
subtype Agent = Vis | Emp | Grd

subtype ConnType = Secure | Insecure | HVAC
subtype TypeConn = Secure | Insecure

Processes = {0..2}
MaxAgents = 5
AgentCnt = {0..MaxAgents}

-- events (first-order and higher order)
channel count : Countable.AgentCnt                              -- local state
channel goto : Agent.Room.Room          -- agents' movement



channel ap,hvac,rA,c2,eaR : Processes                           --locations

-- Processes for the restoration area
R0 = (goto!Vis!RA!Corr2 -> R0) [] (goto!Vis!Corr2!RA -> R0)
R1 = SKIP


-- Count Process (state) for the restoration area
R(v) = (v < MaxAgents & goto!Vis!Corr2!RA -> R(v+1))
               [] (v > 0 & goto!Vis!RA!Corr2 -> R(v-1))


-- Adaptation Function for the restoration area
Pi_RA = (goto!Grd!Str!Corr2 ->  rA!0 -> Pi_RA)
        [] (goto!Grd!Corr2!Str -> rA!1 -> Pi_RA)
        [] (goto!Vis!Corr2!RA -> Pi_RA) [] (goto!Vis!RA!Corr2 -> Pi_RA)

--Adaptable Process for the restoration area
adtProc = let map = \ id @ (if id == 0 then R0
                                              else if id == 1 then R1
                                              else SKIP)
               within
                       (rA?id : Processes -> (map(id) /\ adtProc))

-- Composed Process for the restoration area
ResArea = let
                       adtMgr = (R(0) [|{|goto.Vis|}|] Pi_RA)
                       proc = ((R1 /\ adtProc) [| {| goto.Vis,rA |} |] adtMgr) \ {| rA |}
               within
                       normal(proc)

-- Processes for Corridor 2
C2_0 =  (goto!Vis!Str!Corr2 -> C2_0)
               [] (goto!Vis!Corr2!Str -> C2_0)
               [] (goto!Grd!Corr2!Str -> C2_0)
```

```
                [] (goto!Vis!RA!Corr2 -> C2_0)
                [] (goto!Vis!Corr2!RA -> C2_0)
                [] (goto!Grd!Str!Corr2 -> C2_0)


C2_1 =     (goto!Vis!Str!Corr2 -> C2_1)
                [] (goto!Vis!Corr2!Str -> C2_1)
                [] (goto!Grd!Str!Corr2 -> C2_1)
                [] (goto!Vis!RA!Corr2 -> C2_1)
                [] (goto!Vis!Corr2!RA -> C2_1)

-- Count process for corridor 2
C(v,va,g) = (v + va < 5 & goto!Vis!Str!Corr2 -> C(v+1,va,g))
                [] (v > 0 & goto!Vis!Corr2!Str -> C(v-1,va,g))
                [] (v > 0 and va < 5 & goto!Vis!Corr2!RA -> C(v-1,va+1,g))
                [] (v < 5 and va > 0 & goto!Vis!RA!Corr2 -> C(v+1,va-1,g))
                [] (count!Vis!v+va -> C(v,va,g))
                [] (g >0 & goto!Grd!Corr2!Str -> C(v,va,g-1))
                [] (g<1 & goto!Grd!Str!Corr2 -> C(v,va,g+1))

-- Adpation Function for corridor 2
Pi_C2 =         (goto!Vis!Str!Corr2 -> count!Vis?v : AgentCnt -> if v == 1 then c2!1 -> Pi_C2 else Pi_C2)
                [] (goto!Vis!Corr2!Str -> count!Vis?v : AgentCnt -> if v == 0 then c2!0 -> Pi_C2 else Pi_C2)
                [] (goto!Grd?_?_ -> Pi_C2) [] (goto!Vis!Corr2!RA -> Pi_C2) [] (goto!Vis!RA!Corr2 -> Pi_C2)

--Adaptable Process for corridor 2
adtProc_C2 = let map = \ id @ (if id == 0 then C2_0
                                          else if id == 1 then C2_1
                                          else SKIP)
                within
                        (c2?id : Processes -> (map(id) /\ adtProc_C2))

-- Composed Process for corridor 2
CorrTwo = let
                        adtMgr = (C(0,0,0) [|{|goto,count|}|] Pi_C2) \ {|count|}
                        nAdtMgr = normal(adtMgr)
                        proc = ((C2_0 /\ adtProc_C2) [| {| goto,c2 |} |] nAdtMgr) \ {| c2 |}
                        nproc = normal(proc)
                within
                        nproc



-- System
Implement1 = (CorrTwo [| {| goto.Grd, goto.Vis.Corr2.RA,goto.Vis.RA.Corr2 |} |] ResArea)

--Assertion
Specification = (goto!Grd!Str!Corr2 -> temp)
                                [] (goto!Vis!Str!Corr2 -> Specification)
                                [] (goto!Vis!Corr2!Str -> Specification)
temp = (goto!Vis?x:Room?_:{y | y <- Room, x != y} -> temp)
                [] (goto!Grd!Corr2!Str -> Specification)

assert Specification [T= Implement1

-- Assertion 2
GuardLeave(n) = (goto!Grd!Str!Corr2 -> tempGrdLeave(0,n))
```

```
                              [] (n < MaxAgents & goto!Vis!Str!Corr2 -> GuardLeave(n+1))
                              [] (n > 0 & goto!Vis!Corr2!Str -> GuardLeave(n-1))

tempGrdLeave(n,m) = (n == 0 & goto!Grd!Corr2!Str -> GuardLeave(m))
                [] (n < MaxAgents and m > 0 & goto!Vis!Corr2!RA -> tempGrdLeave(n+1,m-1))
                [] (n > 0 and m < MaxAgents & goto!Vis!RA!Corr2 -> tempGrdLeave(n-1,m+1))
                [] (m > 0 & goto!Vis!Corr2!Str -> tempGrdLeave(n,m-1))
                [] (m < MaxAgents & goto!Vis!Str!Corr2 -> tempGrdLeave(n,m+1))

assert GuardLeave(0) [T= Implement1
```

E FDR CODE FOR VERIFYING REQUIREMENT 2

```
transparent normal

datatype Countable = Vis | Emp | Grd | Secure | Insecure | HVAC

datatype Room = EA | EB | EC | ED | Corr
subtype Agent = Vis | Emp | Grd

subtype ConnType = Secure | Insecure | HVAC
subtype TypeConn = Secure | Insecure

Processes = {0..2}
MaxAgents = 3
AgentCnt = {0..MaxAgents}

-- events (first-order and higher order)
channel count : Countable.AgentCnt                    -- local state
channel goto : Room.Room                              -- agents' movement
channel inc,dec : Room
channel max : AgentCnt

channel ap,hvac,rA,c2,eaR : Processes                         --locations

-- helper functions
notR(rm) = {x | x <- Room, x != rm}

-- The allowed actions
Movement = (goto!EA!EB -> Movement) [] (goto!EB!EA -> Movement)
                        [] (goto!EB!ED -> Movement) [] (goto!ED!EB -> Movement)
                        [] (goto!ED!EC -> Movement) [] (goto!EC!Corr -> Movement)
                        [] (goto!EA!Corr -> Movement) [] (goto!Corr!EA -> Movement)


-- Processes for a room in the exhibition area
R_0(rm) = (goto!rm?r : notR(rm) -> R_0(rm)) [] (goto?r : notR(rm)!rm -> R_0(rm))
R_1(rm) = SKIP

-- Count Process (state) for a room in the exhibition area
C_EA(rm,n) = (n < MaxAgents & goto?_!rm -> C_EA(rm,n+1))
                        [] (n > 0 & goto!rm?_ -> C_EA(rm,n-1))
                        [] count!Vis!n -> C_EA(rm,n)

-- Maximum Agents for a room in the exhibition area
M(rm,n) = (max!n -> M(rm,n))
                [] (inc!rm -> n < MaxAgents & M(rm,n+1))
```

```
                [] (dec!rm -> n > 0 & M(rm,n-1))

-- Adaptation Function for a room in the exhibition area
Pi_EA(rm) = goto?_!rm -> count!Vis?n -> max?m
                                -> (if n == m then eaR!0 -> Pi_EA(rm)
                                        else if n > m then eaR!1 ->  Pi_EA(rm)
                                        else Pi_EA(rm))
                [] goto!rm?_ -> count!Vis?n -> max?m
                                -> (if n == m - 1 then eaR!0 -> Pi_EA(rm)
                                        else if n >= m then eaR!2  ->  Pi_EA(rm)
                                        else Pi_EA(rm))

--Adaptable Process for a room in the exhibition area
adtProc_EA(rm) =
                let
                        adjacentRooms = (| EA => EB, EB => ED, ED => EC, EC => EA |)
                        adjRm = mapLookup(adjacentRooms,rm)
                        map = \ id @ (if id == 0 then R_0(rm)
                                                else if id == 1 then dec!adjRm -> R_1(rm)
                                                else if id == 2 then inc!adjRm -> R_1(rm)
                                                else SKIP)
                within
                        (eaR?id : Processes -> (map(id) /\ adtProc_EA(rm)))

-- Composed Process for a room in the exhibition area
Room_EA(rm) = let
                        countProc = C_EA(rm,0) [|{|goto|}|] Movement
                        adtMgr = (Pi_EA(rm) [| {|count,goto|}|] countProc) \ {|count|}
                        MaxAdtMgr = (adtMgr [|{|max|}|] M(rm,MaxAgents)) \ {| max |}
                        nAdtMgr = normal(MaxAdtMgr)
                        proc = ((R_0(rm) /\ adtProc_EA(rm)) [| {| goto,eaR |} |] nAdtMgr) \ {| eaR |}
                within
                        normal(proc)



ExhibitionArea =
                                let
                                        AB = Room_EA(EA) [| { goto.EA.EB,goto.EB.EA, inc.EB,dec.EB }|] Room_EA(EB)
                                        CD = Room_EA(EC) [| {goto.EC.ED,goto.ED.EC, inc.ED,dec.ED}|] Room_EA(ED)
                                within
                                        (AB [| {|goto.EB.ED,goto.ED.EB, inc , dec|}|] CD) \ {|inc,dec|}

-- Exhibition Area assertion
channel violation
Specification(n) =
                 (n < MaxAgents*12 & goto!Corr?_  -> Specification(n+1))
                [] (n > 0  & goto?_!Corr -> Specification(n-1))
                [] (goto?_:notR(Corr)?_:notR(Corr) -> Specification(n))


assert  Specification(0)  [T= ExhibitionArea

RoomSpec(rm,n) =
                                (n > 0 & goto!rm?_ -> RoomSpec(rm,n-1))
                                [] (n < MaxAgents & goto?_!rm -> RoomSpec(rm,n+1))
```

```
assert RoomSpec(EA,0) [T= Room_EA(EA) \ {|dec,inc|}
```

F FDR CODE FOR VERIFYING REQUIREMENT 3

```
transparent normal

datatype Countable = Vis | Emp | Grd | Secure | Insecure | HVAC

datatype Room = Corr2 | Str | Corr
subtype Agent = Vis | Emp | Grd

subtype ConnType = Secure | Insecure | HVAC
subtype TypeConn = Secure | Insecure

Processes = {0..5}
MaxAgents = 3
AgentCnt = {0..MaxAgents}

-- events (first-order and higher order)
channel count : Bool.AgentCnt.Bool                    -- local state
channel goto : Agent.Room.Room                        -- agents' movement
channel connect,disconnect : ConnType          -- connect/disconnect


channel ap,hvac,rA,c2,eaR : Processes                          --locations

-- Processes for the access point
AP_0 = (connect?t : TypeConn -> AP_0) [] (disconnect?t:TypeConn -> AP_0)
AP_1 = (connect!Secure -> AP_1) [] (disconnect!Secure -> AP_1)

-- Count Process (state) for the access point
C_AP(v,e,h) =   (v < MaxAgents & connect!Insecure -> C_AP(v+1, e,h))
                      [] (e < MaxAgents & connect!Secure -> C_AP(v, e+1,h))
                      [] (e > 0 & disconnect!Secure -> C_AP(v,e-1,h))
                      [] (v > 0 & disconnect!Insecure -> C_AP(v-1,e,h))
                      [] (if v > 0 then count!false!v!h -> C_AP(v,e,h) else count!true!e!h -> C_AP(v,e,h))
                      [] (connect!HVAC -> C_AP(0,e,true)) [] (disconnect!HVAC -> C_AP(v,e,false))

-- Adaptation Function for the access point
Pi_AP =         (connect!Secure          -> count?sec : Bool?c : AgentCnt?h : Bool -> if not sec then ap!1 -> Pi_AP

                              else if not h then ap!5 -> Pi_AP

                              else if h and c == 1 then ap!3 -> Pi_AP

                              else Pi_AP)
                [] (disconnect!Insecure -> count?sec : Bool?c : AgentCnt?h : Bool -> if not h and c == 0 then ap!0 -> Pi_AP
     else Pi_AP)
                [] (connect!Insecure    -> count?sec : Bool?c : AgentCnt?h : Bool -> if h then ap!2 -> Pi_AP else Pi_AP)
                [] (disconnect!Secure   -> count?sec : Bool?c : AgentCnt?h : Bool -> if sec and c == 0 then ap!4 -> Pi_AP
     else Pi_AP)

-- Adaptable Process for the access point
adtProc_AP = let map = \ id @ (if id == 0 then connect!HVAC -> AP_0
                                          else if id == 1 then disconnect!Insecure -> connect!HVAC -> AP_1
                                          else if id == 2 then disconnect!HVAC -> AP_0
                                          else if id == 3 then AP_1
```

```
                                        else if id == 5 then connect!HVAC -> AP_1
                                        else if id == 4 then AP_0
                                        else SKIP)
                within
                        (ap?id : Processes -> (map(id) /\ adtProc_AP))

-- Composed Process for access point
AccessPt = let
                        connections ={|connect.Insecure,connect.Secure,disconnect.Insecure,disconnect.Secure, count |}
                        adtMgr = (C_AP(0,0,false) [|  connections  |] Pi_AP) \ {|count|}
                        nAdtMgr = normal(adtMgr)
                        proc = (((connect!HVAC -> AP_0) /\ adtProc_AP) [| {|connect, disconnect, ap|} |] nAdtMgr) \ {| ap |}
                within
                        normal(proc)

-- Processes for the HVAC
Conn = SKIP
Dis = SKIP

-- Adaptation Function for the HVAC
Pi_H =  (connect!HVAC -> hvac!0 -> Pi_H )
                [] (disconnect!HVAC -> hvac!1 -> Pi_H)

-- Adaptable Process for the HVAC
adtProc_H = let map = \ id @ (if id == 0 then Conn
                                        else if id == 1 then Dis
                                        else SKIP)
                within
                        (hvac?id : Processes -> (map(id) /\ adtProc_H))

-- Composed Process for HVAC
HVACComp = let
                        proc = ((Conn /\ adtProc_H) [| {|hvac|} |] Pi_H) \ {| hvac |}
                within
                        normal(proc)

-- Process for Stairs

Stairs(v,e) =   (v > 0 & goto!Vis!Corr2!Str -> disconnect!Insecure -> Stairs(v - 1,e))
                                [] (v > 0 & goto!Vis!Str!Corr -> Stairs(v,e))
                                [] (e > 0 & goto!Emp!Corr2!Str -> disconnect!Secure -> Stairs(v,e-1))
                                [] (e > 0 & goto!Emp!Str!Corr -> Stairs(v,e- 1))
                                [] (v < MaxAgents & goto!Vis!Str!Corr2 -> connect!Insecure -> Stairs(v + 1,e))
                                [] (v < MaxAgents & goto!Vis!Corr!Str -> Stairs(v + 1, e))
                                [] (e < MaxAgents & goto!Emp!Str!Corr2 -> connect!Secure->Stairs(v ,e + 1))
                                [] (e < MaxAgents & goto!Emp!Corr!Str-> Stairs(v,e + 1))

-- Composition of Components
Implementation3 = (HVACComp [| {|connect.HVAC, disconnect.HVAC |} |] AccessPt)
--[| {|connect.Secure,connect.Insecure,disconnect.Secure,disconnect.Insecure |}|] Stairs(0,0)

-- Assertion for Requirement 3
Requirement3 =  connect!HVAC -> Requirement_2(0)
Requirement_1(c) =              (c > 0 and c < MaxAgents & connect!Insecure -> Requirement_1(c+1))
                                        [] (c == 0 & connect!Insecure -> disconnect!HVAC -> Requirement_1(c+1))
                                        [] (c > 1 & disconnect!Insecure -> Requirement_1(c-1))
                                        [] (c == 1 & disconnect!Insecure -> connect!HVAC -> Requirement_1(c-1))
```

```
                                        [] (c > 0 & connect!Secure -> disconnect!Insecure -> connect!HVAC -> Requirement_2(1))
                                        [] (c == 0 & connect!Secure  -> Requirement_2(1))
Requirement_2(c) = (c < MaxAgents & connect!Secure -> Requirement_2(c+1))
                                        [] (c > 0 & disconnect!Secure -> Requirement_2(c-1))
                                        [] ( c == 0 & (connect!Insecure -> disconnect!HVAC -> Requirement_1(1)))


assert Requirement3 [T= Implementation3
```