# A REMAINING DEFINITIONS AND PROOFS

## A.1 Remaining Meta-functions

$\texttt{contract\_for}(\sigma, b) = (n, \checkmark)$ if $\exists ee \mapsto \widetilde{l}\checkmark \in \texttt{blame}_\sigma(\sigma)$. where $\texttt{ep}(ee) = b.y \wedge \texttt{nm}(ee) = n$

$\texttt{contract\_for}(\sigma, b) = (n, \times)$ if $(\exists ee \mapsto \widetilde{l}\times \in \texttt{blame}_\sigma(\sigma)$. where $\texttt{ep}(ee) = b.y \wedge \texttt{nm}(ee) = n)$
$\wedge\ (\forall ee' \mapsto \widetilde{l'}c \in \texttt{blame}_\sigma(\sigma).$ if $\texttt{ep}(ee') = b$
then $\texttt{nm}(ee) = k$ and $c = \times)$

$\texttt{ep}(a.r\ \texttt{satisfies}\ n\langle \texttt{v} \rangle) = a.r$
$\texttt{ep}(a.i\ \texttt{expects}\ se) = a.i$
$\texttt{nm}(a.r\ \texttt{satisfies}\ n\langle \texttt{v} \rangle) = n$
$\texttt{nm}(a.i\ \texttt{expects}\ se) = \texttt{nm}(se)$

*names metafunction.*

$$\frac{}{\texttt{names}(\boxed{a}) = \{a\}}$$

$$\frac{}{\texttt{names}(\texttt{mon}^l(\sigma, P^a)) = \{a\}}$$

$$\frac{}{\texttt{names}(m\ \texttt{to}\ a) = \emptyset} \qquad \frac{}{\texttt{names}(\hat{m}\ \texttt{to}\ a) = \emptyset}$$

$$\frac{\widetilde{a_1} = \texttt{names}(P) \qquad \widetilde{a_2} = \texttt{names}(Q)}{\texttt{names}(P \parallel Q) = \widetilde{a_1} \cup \widetilde{a_2}}$$

*labels metafunction.*

$$\frac{}{\texttt{labels}(\boxed{a}) = \emptyset}$$

$$\frac{}{\texttt{labels}(\texttt{mon}^l(\sigma, P^a)) = \{l\}}$$

$$\frac{}{\texttt{labels}(m\ \texttt{to}\ a) = \emptyset} \qquad \frac{}{\texttt{labels}(\hat{m}\ \texttt{to}\ a) = \emptyset}$$

$$\frac{\widetilde{l_1} = \texttt{labels}(P) \qquad \widetilde{l_2} = \texttt{labels}(Q)}{\texttt{labels}(P \parallel Q) = \widetilde{l_1} \cup \widetilde{l_2}}$$

## A.2 Congruence Relation

$$\frac{}{P = P} \qquad \frac{}{P \parallel Q = Q \parallel P} \qquad \frac{P = P' \qquad Q = Q'}{P \parallel Q = P' \parallel Q'}$$

### A.3 Well-formedness

$$\frac{\forall (a \;\texttt{satisfies}\; k\langle v\rangle \mapsto \checkmark) \in C.\; C(a) = k}{C \preceq C}$$

$$\frac{P \vdash_S^C P}{\vdash_S^C P \;\mathsf{wf}}$$

$$\frac{P \vdash_S^C Q \qquad P \vdash_S^C R \qquad \texttt{names}(Q) \cap \texttt{names}(R) = \emptyset \qquad \texttt{labels}(Q) \cap \texttt{labels}(R) = \emptyset}{P \vdash_S^C Q \parallel R}$$

$$\frac{\texttt{conf}_\sigma \preceq C \qquad l \neq \dagger \qquad S = \texttt{spec}(\sigma) \qquad \forall se.\; se \in dom(\texttt{blame}_\sigma) \text{ iff } se \in dom(\texttt{prov}_\sigma) \qquad \forall se \mapsto \widetilde{l} \in \texttt{blame}_\sigma,\; l \in \widetilde{l}.\; P \Vdash \textbf{blame } l \textbf{ for } se \qquad \forall \#n \;\texttt{from}\; b \;\texttt{expects}\; se \mapsto \widetilde{l} \in \texttt{blame}_\sigma,\; l \in \widetilde{l}.\; P \Vdash \textbf{blame } l \textbf{ for } se \qquad \forall a \mapsto n \in C.\; a \;\texttt{satisfies}\; k\langle \texttt{v}\rangle \mapsto \checkmark \in \texttt{conf}_\sigma}{P \vdash_S^C \texttt{mon}^l(\sigma, P^a)}$$

$$\frac{}{P \vdash_S^C \boxed{a}} \qquad \frac{m = \texttt{req}\;\#n\;\texttt{from}\;b\;:\;s\;\texttt{to}\;a \text{ or } m = \texttt{reply}\;\#n\;\texttt{from}\;b\;:\;s\;\texttt{to}\;a}{P \vdash_S^C m\;\texttt{to}\;a}$$

$$\frac{k = C(a) \qquad \{\texttt{identified}(m) \mapsto c\} \preceq C \qquad \forall l \in \widetilde{l}.\; P \Vdash \textbf{blame } l \textbf{ for } \texttt{from}(a) \;\texttt{satisfies}\; k\langle \texttt{index}(m)\rangle \qquad \forall l \in \widetilde{l_{id}}.\; P \Vdash \textbf{blame } l \textbf{ for } \texttt{identified}(m)}{P \vdash_S^C m \;\texttt{with}\; \{\texttt{se-blame}{:=}\widetilde{l};\; \texttt{id-conf}{:=}c;\; \texttt{id-blame}{:=}\widetilde{l_{id}}\}\;\texttt{to}\;b}$$

First, note that we define well-formedness judgment $P \vdash_S^C P'$, where $P$ is the entire process, and $P'$ is some subprocess of $P$. For brevity, we used wellformedness judgment $\vdash_S^C P$ wf where $P$ was the entire process, and so treat that as equivalent to $P \vdash_S^C P$.

Lemma A.1 (Preservation of Well-formedness over congruence). *If $P \vdash_S^C P'$ and $P = P''$ and $P' = P'''$ then $P'' \vdash_S^C P'''$.*

Proof. The congruence relation only changes the order of the processes. The well-formedness relation is not sensitive to ordering of the processes. □

Lemma A.2 (`lift` Preserves Store Well-formedness). If $\mathcal{P}[\mathtt{mon}^l(\sigma, P^a)] \vdash_S^C \mathtt{mon}^l(\sigma, P^a)$ and $\sigma', m$ `with` $\{$`se-blame:=`$\widetilde{l}$; `id-conf:=`$c$; `id-blame:=`$\widetilde{l_{id}}\}$ `to` $b = \mathtt{lift}_\sigma(k, c, m, l)$ and (if $\mathtt{from}(m) \neq a$ then $l = \dagger$), then

$$\forall se \mapsto \widetilde{l} \in \mathtt{blame}_{\sigma'},\ l' \in \widetilde{l}.\ \mathcal{P}[\mathtt{mon}^{l'}(\sigma, P^a)] \Vdash \textbf{blame } l' \textbf{ for } se \text{ and}$$
$$\forall (\#n\ \mathtt{from}\ c\ \mathtt{expects}\ se) \mapsto \widetilde{l} \in \mathtt{blame}_{\sigma'},\ l' \in \widetilde{l}.\ \mathcal{P}[\mathtt{mon}^{l'}(\sigma, P^a)] \Vdash \textbf{blame } l' \textbf{ for } se$$

.

Proof. By inspection of the `lift` rule given, there are two entries that can be updated: the service endpoint, and $se_{id}$. We first show $\forall l' \in \widetilde{l_s}.\ \mathcal{P}[\mathtt{mon}^l(\sigma', P^a)] \Vdash \textbf{blame } l' \textbf{ for } se$, then $\forall l' \in \widetilde{l_{id}}.\ \mathcal{P}[\mathtt{mon}^l(\sigma', P^a)] \Vdash \textbf{blame } l' \textbf{ for } se_{id}$.

For the well-formedness of $se$, we take cases on if $se \in dom(\mathtt{blame}_\sigma)$:

- **Case** $se \notin dom(\mathbf{blame}_\sigma)$:
  It is the case then that $\widetilde{l_s} = \{l\}$. Due to well-formedness, we have that

  $$\forall se.\ se \in dom(\mathtt{blame}_\sigma) \text{ iff } se \in dom(\mathtt{prov}_\sigma)$$

  It will be the case then that $se \notin dom(\mathtt{prov}_\sigma)$ so $pe_s = b$ **intro**. If $b = a$ we can form the following derivation:

  $$\frac{\mathtt{prov}_{\sigma'}(se) = a \textbf{ intro}}{\mathcal{P}[\mathtt{mon}^l(\sigma', P^a)] \Vdash \textbf{blame } l \textbf{ for } se}$$

  Otherwise we are in a partially deployed case (based on the reduction rules that call `lift`). As a result, in these cases $l_a = \dagger$. So we can form the partial deployment derivation:

  $$\frac{\mathtt{prov}_{\sigma'}(se) = b \textbf{ intro}}{\mathcal{P}[\mathtt{mon}^l(\sigma', P^a)] \Vdash \textbf{blame } \dagger \textbf{ for } se}$$

- **Case** $se \in dom(\mathbf{blame}_\sigma)$:
  Due to well-formedness, $\forall se \in \mathtt{blame}_\sigma$, then $\forall l' \in \widetilde{l_s}.\ \mathcal{P}[\mathtt{mon}^l(\sigma, P^a)] \Vdash \textbf{blame } l' \textbf{ for } se$. Additionally, since no provenance or blame or provenance information is overwritten the derivation must hold for $\sigma'$, so, $\forall l' \in \widetilde{l_s}.\ \mathcal{P}[\mathtt{mon}^l(\sigma', P^a)] \Vdash \textbf{blame } l' \textbf{ for } re$.

Next we show the well-formedness of $se_{id}$. We also perform a case analysis on the type of the message together with if $se_{id} \in dom(\mathtt{blame}_\sigma)$.

- **Case** $\mathtt{type}(m) == \mathtt{req} \wedge se_{id} \in dom(\mathbf{blame}_\sigma)$: Due to well-formedness of the store, we have that $\forall se \mapsto \widetilde{l} \in \mathtt{blame}_\sigma,\ l \in \widetilde{l}.\ \mathcal{P}[\mathtt{mon}^l(\sigma, P^a)] \Vdash \textbf{blame } l \textbf{ for } se$, so it must be the case that $\forall l' \in \widetilde{l_{id}}.\ \mathcal{P}[\mathtt{mon}^l(\sigma, P^a)] \Vdash \textbf{blame } l' \textbf{ for } se_{id}$. As a result, since the entry is replaced with itself along with its provenance (i.e., no information is replaced), the well-formedness condition still holds.
- **Case** $\mathtt{type}(m) == \mathtt{req} \wedge se_{id} \notin dom(\mathbf{blame}_\sigma)$: It is the case then that $\widetilde{l_{id}} = \{l\}$. Due to well-formedness, we have that $\forall se.\ se \in dom(\mathtt{blame}_\sigma)$ iff $se \in dom(\mathtt{prov}_\sigma)$. As a result $se_{id} \notin dom(\mathtt{prov}_\sigma)$.

Next we consider if $b = a$. It will be the case that $\texttt{prov}_{\sigma'}(se_{id}) = a$ **intro**. So the derivation for this case is:

$$\frac{\texttt{prov}_{\sigma'} = a \textbf{ intro}}{\mathcal{P}[\texttt{mon}^l(\sigma', P^a)] \Vdash \textbf{blame } l \textbf{ for } se_{id}}$$

Otherwise, if $b \neq a$, we are in a partially deployed case (based on the reduction rules that call `lift`). As a result, in these cases $l_a = \dagger$. So we can form the partial deployment derivation:

$$\frac{\texttt{prov}_{\sigma'} = b \textbf{ intro}}{\mathcal{P}[\texttt{mon}^l(\sigma', P^a)] \Vdash \textbf{blame } \dagger \textbf{ for } se_{id}}$$

- **Case** $\texttt{type}(m) == \texttt{reply} \wedge se_{id} \in dom(\texttt{blame}_\sigma)$:
  The same reasoning applies for the case if $\texttt{type}(m) == \texttt{req}$ and $se_{id} \in dom(\texttt{blame}_\sigma)$ as the entry is looked up in the same fashion for requests and replies.
- **Case** $\texttt{type}(m) == \texttt{reply} \wedge se_{id} \notin dom(\texttt{blame}_\sigma)$:
  It is the case then that $\widetilde{l_{id}} = \widetilde{l_s}$. From well-formedness of the store, $\forall l' \in \widetilde{l_s}.\ \mathcal{P}[\texttt{mon}^l(\sigma, P^a)] \Vdash$ **blame** $l'$ **for** $se$. Also from store well-formedness, it must be the case that $se_{id} \notin dom(\texttt{blame}_\sigma)$ as $\forall se.\ se \in dom(\texttt{blame}_\sigma)$ iff $se \in dom(\texttt{prov}_\sigma)$. Because of this, we know that $\texttt{prov}_{\sigma'}(se_{id}) = se$ **intro**.
  We can directly set up the derivation for an unconfirmed service identification for each $l' \in \widetilde{l_s}$:

$$\frac{\texttt{prov}_{\sigma'}(se_{id}) = (se \textbf{ intro}) \qquad \frac{\textit{(from well-formedness)}}{\mathcal{P}[\texttt{mon}^l(\sigma, P^a)] \Vdash \textbf{blame } l' \textbf{ for } se}}{\mathcal{P}[\texttt{mon}^l(\sigma, P^a)] \Vdash \textbf{blame } l' \textbf{ for } se_{id}}$$

□

Lemma A.3 (`lower` Preserves Store Well-formedness). If
$P \vdash^C_S \texttt{mon}^l(\sigma', P^a) \parallel m \texttt{ with } \{\texttt{se-blame:=}\widetilde{l}\texttt{; id-conf:=}c\texttt{; id-blame:=}\widetilde{l_{id}}\} \texttt{ to } b$ and
$\sigma', m = \texttt{lower}_\sigma(k, c, m \texttt{ with } \{\texttt{se-blame:=}\widetilde{l}\texttt{; id-conf:=}c\texttt{; id-blame:=}\widetilde{l_{id}}\} \texttt{ to } b)$ then
$\forall se \mapsto \widetilde{l} \in \texttt{blame}_{\sigma'},\ l' \in \widetilde{l}.\ \mathcal{P}[\texttt{mon}^{l'}(\sigma, P^a)] \Vdash$ **blame** $l'$ **for** $se$ and
$\forall (\#n \texttt{ from } c \texttt{ expects } se) \mapsto \widetilde{l} \in \texttt{blame}_{\sigma'},\ l' \in \widetilde{l}.\ \mathcal{P}[\texttt{mon}^{l'}(\sigma, P^a)] \Vdash$ **blame** $l'$ **for** $se$
.

Proof. Since the message is well-formed, there will be blame consistent with provenance for the reply and service identified. Additionally, the provenance information in the receiving adapter is not overwritten so any existing derivations of blame consistent with provenance will still hold. As a result, the services added to the store will still have a valid derivation of blame consistent with provenance. □

Lemma A.4 (`lift` and `lower` preserve state). If $\mathcal{P}[\texttt{mon}^l(\sigma, P^a)] \vdash^C_S \texttt{mon}^l(\sigma', P^a)$ and either $\sigma', \hat{m} = \texttt{lift}_\sigma(k, c, m, l)$ or $\sigma', m = \texttt{lower}_\sigma(k, c, \hat{m})$, then
(1) $S = \texttt{spec}(\sigma')$
(2) $\forall se.\ se \in dom(\texttt{blame}_{\sigma'})$ iff $se \in dom(\texttt{prov}_{\sigma'})$
(3) $\forall a \mapsto k \in C.\ a \texttt{ satisfies } k\langle \texttt{v} \rangle \mapsto \checkmark \in \texttt{conf}_{\sigma'}$

Proof. For the first case, $S = \texttt{spec}(\sigma')$, it is clear from the store updates in the metafunctions that the store is not changed.

For the second case, $\forall se.\ se \in dom(\texttt{blame}_{\sigma'})$ iff $re \in dom(\texttt{prov}_{\sigma'})$, it is clear by inspection that the updates to the blame registry and provenance registry affect the same domain of services.

For the third case, $\forall a \mapsto n \in C.\ a \texttt{ satisfies } k\langle \texttt{v} \rangle \mapsto \checkmark \in \texttt{conf}_{\sigma'}$, services added to the blame registry do not ever become unconfirmed after becoming confirmed (i.e., note the usage and

definition of the `confirmed_or` function). As a result, if $\sigma$ had this property, then no replacement could make the service become unconfirmed so it will still hold for $\sigma'$.

□

Theorem A.5 (Preservation of Well-formedness). *If* $P \vdash_S^C P$ $P \rightarrow P'$ *then* $P' \vdash_S^C P'$.

Proof. By case analysis on step used.

- **Case Black-box Send Request**
  From the definition of the rule, we have

$$\frac{n \text{ fresh} \qquad a \neq b}{\mathcal{P}[\boxed{a}] \longrightarrow \mathcal{P}[\boxed{a} \parallel \texttt{request \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b]}$$

  where $P = \mathcal{P}[\boxed{a}]$ and $P' = \mathcal{P}[\boxed{a} \parallel \texttt{request \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b]$. We also have from our congruence rules that

$$\mathcal{P}[\boxed{a} \parallel \texttt{request \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b] = \mathcal{P}[\boxed{a}] \parallel \texttt{request \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b$$

  We also have that $P \vdash_S^C P$. We can re-use the proof derivation of that to construct $P' \vdash_S^C \mathcal{P}[\boxed{a}]$. $P'$ can be used in place of $P$ on the left-hand side as it contains the same relevant processes to the blame-consistent-with-provenance relation. We can now construct a new derivation for $P'$. We can use the base message well-formed rule along with the composition rule to get:

$$\frac{\dfrac{\dots}{P' \vdash_S^C \mathcal{P}[\boxed{a}]} \qquad \dfrac{}{P' \vdash_S^C \texttt{request \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b} \quad \begin{array}{c}\texttt{names}(\mathcal{P}[\boxed{a}]) \cap \texttt{names}(\texttt{request \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b) = \emptyset \\ \texttt{labels}(\mathcal{P}[\boxed{a}]) \cap \texttt{labels}(\texttt{request \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b) = \emptyset\end{array}}{P' \vdash_S^C \mathcal{P}[\boxed{a}] \parallel \texttt{request \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b}$$

- **Case Black-box Send Reply**
  From the definition of the rule, we have

$$\frac{a \neq b}{\mathcal{P}[\boxed{a}] \longrightarrow \mathcal{P}[\boxed{a} \parallel \texttt{reply \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b]}$$

  where $P = \mathcal{P}[\boxed{a}]$ and $P' = \mathcal{P}[\boxed{a} \parallel \texttt{reply \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b]$. We also have from our congruence rules that

$$\mathcal{P}[\boxed{a} \parallel \texttt{reply \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b] = \mathcal{P}[\boxed{a}] \parallel \texttt{reply \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b$$

  . We also have that $P \vdash_S^C P$. We can re-use the proof derivation of that to construct $P' \vdash_S^C \mathcal{P}[\boxed{a}]$. $P'$ can be used in place of $P$ on the left-hand side as it contains the same relevant processes to the blame-consistent-with-provenance relation. We can now construct a new derivation for $P'$. We can use the base message well-formed rule along with the composition rule to get:

$$\frac{\dfrac{\dots}{P' \vdash_S^C \mathcal{P}[\boxed{a}]} \qquad \dfrac{}{P' \vdash_S^C \texttt{reply \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b} \quad \begin{array}{c}\texttt{names}(\mathcal{P}[\boxed{a}]) \cap \texttt{names}(\texttt{reply \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b) = \emptyset \\ \texttt{labels}(\mathcal{P}[\boxed{a}]) \cap \texttt{labels}(\texttt{reply \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b) = \emptyset\end{array}}{P' \vdash_S^C \mathcal{P}[\boxed{a}] \parallel \texttt{reply \#}n \texttt{ from } a \texttt{ containing } s \texttt{ to } b}$$

- **Case Black-box Receive**
  From the definition of the rule, we have

$$\frac{}{\mathcal{P}[\boxed{a} \parallel m \texttt{ to } a] \rightarrow \mathcal{P}[\boxed{a}]}$$

where $P = \mathcal{P}[\boxed{a} \parallel m \texttt{ to } a]$ and $P' = \mathcal{P}[\boxed{a}]$. We also have that $P \vdash^C_S P$. We can re-use the proof derivation of that to construct $P' \vdash^C_S \mathcal{P}[\boxed{a}]$ by removing the part of the derivation that included the rule used for the received message. $P'$ can be used in place of $P$ on the left-hand side as it contains the same relevant processes to the blame-consistent-with-provenance relation. With this, we can now construct a new derivation for $P'$ directly to get $P' \vdash^C_S \mathcal{P}[\boxed{a}]$.

- **Case Adapter Send Enhanced**
  From the definition of the rule, we have

$$\frac{(k, \checkmark) = \texttt{contract\_for}_\sigma(b, m) \qquad \sigma', \hat{m} = \texttt{lift}_\sigma(k, \checkmark, m, l)}{\mathcal{P}[\texttt{mon}^l(\sigma, P^a) \parallel m \texttt{ to } b)] \rightarrow \mathcal{P}[\texttt{mon}^l(\sigma', P^a) \parallel \hat{m} \texttt{ to } b]}$$

  where $P = \mathcal{P}[\texttt{mon}^l(\sigma, P^a \parallel m \texttt{ to } b)]$ and $P' = \mathcal{P}[\texttt{mon}^l(\sigma', P^a) \parallel \hat{m} \texttt{ to } b]$. From Theorems A.2 and A.4, we can show that $\mathcal{P}[\texttt{mon}^l(\sigma', P^a)] \vdash^C_S \texttt{mon}^l(\sigma', P^a)$. We note that we can use $P'$ in place of $\mathcal{P}[\texttt{mon}^l(\sigma', P^a)]$ as it contains everything but the extra enhanced message, so all derivations will still hold for $P'$. Since the labels came from the store which is well-formed, along with the confirmation status, the blame premises for the message hold. That is, $\forall l \in \widetilde{l}$. $P \Vdash$ **blame** $l$ **for** $b$ $\texttt{satisfies}$ $k\langle\texttt{index}(m)\rangle$ and $\forall l \in \widetilde{l_{id}}$. $P \Vdash$ **blame** $l$ **for** $\texttt{identified}(m)$
  We can now set up the following derivation.

$$\frac{\dfrac{\begin{array}{c}\hat{m} = P \vdash^C_S m \texttt{ with } \{\texttt{se-blame:=}\widetilde{l}\texttt{; id-conf:=}c\texttt{; id-blame:=}\widetilde{l_{id}}\} \texttt{ to } b \\ k = C(b) \qquad \texttt{type}(m) == \texttt{req} \\ \{\texttt{identified}(m) \mapsto c\} \preceq C \qquad \forall l \in \widetilde{l}.\ P \Vdash \textbf{blame } l \textbf{ for } b \texttt{ satisfies } k\langle\texttt{index}(m)\rangle \\ \forall l \in \widetilde{l_{id}}.\ P \Vdash \textbf{blame } l \textbf{ for } \texttt{identified}(m)\end{array}}{P' \vdash^C_S \hat{m} \texttt{ to } b} \qquad \frac{}{P' \vdash^C_S \texttt{mon}^l(\sigma', P^a)}}{\begin{array}{c}P' \vdash^C_S \mathcal{P}[\texttt{mon}^l(\sigma', P^a)] \parallel \hat{m} \texttt{ to } b \\ \texttt{names}(\hat{m} \texttt{ to } b) \cap \texttt{names}(\texttt{mon}^l(\sigma', P^a)) = \emptyset \\ \texttt{labels}(\hat{m} \texttt{ to } b) \cap \texttt{labels}(\texttt{mon}^l(\sigma', P^a)) = \emptyset\end{array}}$$

- **Case Adapter Receive Enhanced**
  From the definition of the rule, we have

$$\frac{(k, \checkmark) = \texttt{contract\_for}_\sigma(b, \hat{m}) \qquad \sigma', m = \texttt{lower}_\sigma(k, \checkmark, m)}{\mathcal{P}[\texttt{mon}^l(\sigma, P^a) \parallel \hat{m} \texttt{ to } a] \rightarrow \mathcal{P}[\texttt{mon}^l(\sigma', P^a \parallel m \texttt{ to } a)]}$$

  where $P = \mathcal{P}[\texttt{mon}^l(\sigma, P^a) \parallel \hat{m} \texttt{ to } a]$ and $P' = \mathcal{P}[\texttt{mon}^l(\sigma', P^a \parallel m \texttt{ to } a)]$. Since the enhanced message is well-formed, we can use Theorems A.3 and A.4 to get that the new adapter state $\sigma'$ is well-formed. Additionally, all unenhanced messages are well-formed, so we can use the parallel composition rule to get

$$\frac{\dfrac{}{P' \vdash^C_S m \texttt{ to } a} \qquad \dfrac{}{P' \vdash^C_S \texttt{mon}^l(\sigma', P^a)}}{\begin{array}{c}P' \vdash^C_S \mathcal{P}[\texttt{mon}^l(\sigma', P^a) \parallel m \texttt{ to } a] \\ \texttt{names}(\texttt{mon}^l(\sigma', P^a)) \cap \texttt{names}(m \texttt{ to } a) = \emptyset \qquad \texttt{labels}(\texttt{mon}^l(\sigma', P^a)) \cap \texttt{labels}(m \texttt{ to } a) = \emptyset\end{array}}$$

- **Case Adapter Send Unenhanced**
  From the definition of the rule, we have

$$\frac{\begin{array}{c}(k, \text{✗}) = \texttt{contract\_for}_\sigma(b, m) \\ \sigma', \hat{m} = \texttt{lift}_\sigma(\text{✗}, k, m, l) \qquad \sigma'', m = \texttt{lower}_{\sigma'}(k, \text{✗}, \hat{m})\end{array}}{\mathcal{P}[\texttt{mon}^l(\sigma, P^a \parallel m \texttt{ to } b)] \rightarrow \mathcal{P}[\texttt{mon}^l(\sigma'', P^a) \parallel m \texttt{ to } b]}$$

where $P = \mathcal{P}[\mathtt{mon}^l(\sigma, P^a \parallel m \ \mathtt{to}\ b)]$ and $P' = \mathcal{P}[\mathtt{mon}^l(\sigma'', P^a) \parallel m \ \mathtt{to}\ b]$. We can use Theorems A.2 to A.4 to get that the new adapter state $\sigma''$ is well-formed. Additionally, all unenhanced messages are well-formed, so we can use the composition rule to get

$$\frac{\dfrac{}{P' \vdash^C_S m \ \mathtt{to}\ b} \qquad \dfrac{}{P' \vdash^C_S \mathtt{mon}^l(\sigma', P^a)}}{\begin{array}{c}\mathtt{names}(\mathtt{mon}^l(\sigma', P^a)) \cap \mathtt{names}(m \ \mathtt{to}\ b) = \emptyset \qquad \mathtt{labels}(\mathtt{mon}^l(\sigma', P^a)) \cap \mathtt{labels}(m \ \mathtt{to}\ b) = \emptyset \\ \hline P' \vdash^C_S \mathcal{P}[\mathtt{mon}^l(\sigma', P^a) \parallel m \ \mathtt{to}\ b]\end{array}}$$

- **Case Adapter Receive Unenhanced**
  From the definition of the rule, we have

$$\frac{\begin{array}{c}(k, c) = \mathtt{contract\_for}_\sigma(a, m) \\ \sigma', \hat{m} = \mathtt{lift}_\sigma(\text{✗}, m, \dagger) \qquad \sigma'', m = \mathtt{lower}_{\sigma'}(k, \text{✗}, \hat{m})\end{array}}{\mathcal{P}[\mathtt{mon}^l(\sigma, P^a) \parallel m \ \mathtt{to}\ a] \rightarrow \mathcal{P}[\mathtt{mon}^l(\sigma'', P^a \parallel m \ \mathtt{to}\ a)]}$$

  where $P = \mathcal{P}[\mathtt{mon}^l(\sigma, P^a) \parallel m \ \mathtt{to}\ a]$ and $P' = \mathcal{P}[\mathtt{mon}^l(\sigma'', P^a \parallel m \ \mathtt{to}\ a)]$. We can use Theorems A.2 to A.4 to get that the new adapter state $\sigma''$ is well-formed. Additionally, all unenhanced messages are well-formed, so we can use the composition rule to get

$$\frac{\dfrac{}{P' \vdash^C_S m \ \mathtt{to}\ a} \qquad \dfrac{}{P' \vdash^C_S \mathtt{mon}^l(\sigma'', P^a)}}{\begin{array}{c}\mathtt{names}(\mathtt{mon}^l(\sigma'', P^a)) \cap \mathtt{names}(m \ \mathtt{to}\ a) = \emptyset \qquad \mathtt{labels}(\mathtt{mon}^l(\sigma'', P^a)) \cap \mathtt{labels}(m \ \mathtt{to}\ a) = \emptyset \\ \hline P' \vdash^C_S \mathcal{P}[\mathtt{mon}^l(\sigma'', P^a \parallel m \ \mathtt{to}\ a)]\end{array}}$$

- **Case Adapter Bypass Send and Receive**
  The adapter state is not changed and all unenhanced messages are well-formed so we can simply use the composition rule to form the new derivation of well-formedness.

With all rules of the reduction relation satisfying preservation of well-formedness, the proof is complete.

□

**Correct Blame.** If well-formed $P_1 = \mathcal{P}_1[\mathtt{mon}^l(\sigma_1, P_1^a)]$ and $P_1 \rightarrow P_2$ and $P_2 = \mathcal{P}_2[\mathtt{mon}^l(\sigma_2, P_2^a)]$ and $\mathtt{errors}_{\sigma_2} = \{le\} \cup \mathtt{errors}_{\sigma_1}$ then

(1) if $le = \mathtt{Pre}(se, l_e)$, then
  (a) if $P_1 = \mathcal{P}[\mathtt{mon}^l(\sigma_1, P^a \parallel m \ \mathtt{to}\ b)]$ and
    $P_2 = \mathcal{P}[\mathtt{mon}^l(\sigma_2, P^a) \parallel m' \ \mathtt{to}\ b]$ then $l_e = l$
  (b) if $P_1 = \mathcal{P}[\mathtt{mon}^l(\sigma_1, P^a) \parallel m \ \mathtt{to}\ a]$ and
    $P_2 = \mathcal{P}[\mathtt{mon}^l(\sigma_2, P^a \parallel m \ \mathtt{to}\ a)]$ then $l_e = \dagger$
(2) if $le = \mathtt{Post}(se, \widetilde{l})$, then $\forall l \in \widetilde{l}.\ P_2 \Vdash$ **blame** $l$ **for** $se$.

Proof. The first case (precondition error) is a direct result of the reduction rules for receiving a message and the `lift` metafunction for contract checking. The second case (postcondition error) is a direct result of well-formedness (specifically $\forall se \mapsto \widetilde{lc} \in \mathtt{blame}_{\sigma'},\ l \in \widetilde{l}.\ P \Vdash$ **blame** $l$ **for** $se$) and the `lift` metafunction for contract checking. □

# B FULL SPECIFICATIONS

## B.1 Evernote

```
service UserStore {
  getNoteStoreUrl(authToken)
  @where index is « authToken »
  @identifies NoteStore at « result » with index « authToken »

  getUser(authToken)
  @where index is « authToken »
  @requires « length(authToken) > 0 »
}
service NoteStore {
  listNotebooks(authToken)
  @where index is « authToken »
  @requires « length(authToken) > 0 »

  listLinkedNotebooks(authToken)
  @where index is « authToken »
  @foreach notebook in « result » identifies NoteStore at « notebook.noteStoreUrl »
    with index « authToken + notebook.shareKey »
  @ensures « for notebook in result: assert(notebook.shareKey != None) »

  getSharedNotebookByAuth(authToken)
  @where index is « authToken »
  @identifies NoteStore at receiver with index « result.authToken »

  authenticateToSharedNotebook(shareKey, authToken)
  @where index is « shareKey + authToken »
  @identifies NoteStore at receiver with index « result.authenticationToken »
  @requires « length(shareKey) > 0 »

  findNotes(authToken, filter, offset, maxNotes)
  @where index is « authToken »
  @requires « offset >= 0 »
  @ensures « result.totalNotes <= maxNotes and result.totalNotes == length(result.notes) »
  @ensures «
    for note in result:
      try: UUID(note.notebookGuid, version=4)
      except ValueError: return False
  »
  @ensures «
    for note in result:
      for resource in note.resources:
        assert (resource.mime in mimetypes.types_map.values())
  »
}
```

### B.2 Twitter

```
from rfc822 import parsedate_tz
service TwitterOAuth {
  /oauth/access_token(req)
  @identifies Twitter at receiver with index
    « 'oauth:' + parse_querystring(result.content).get('oauth_token') »
}

service Twitter {
  /1.1/status/user_timeline.json(request)
  @where index is « 'oauth:' + request['headers'].get('Authorization') »
}

service TwitterTweets {
  /1.1/friends/list.json(request)
  @where index is « 'u' + request['args'].get('user_id', request['args'].get('screen_name')) »
  @requires « 'user_id' in request['args'] or 'screen_name' in request['args']»
  @requires « request['args'].get('count', 0) >= 0 and  'Authorization' in request['headers'] »
  @ensures « type(result['body']['users']) == list »
  @foreach f in « result['body']['users'] » identifies TwitterTweets
    at « request['headers']['Host'] » with index « "u" + f['id_str'] »

  /1.1/statuses/user_timeline.json(request)
  @where index is « 'u' + request['args'].get('user_id', request['args'].get('screen_name')) »
  @requires « 'Authorization' in request['headers'] and \
              ('user_id' in request['args'] or 'screen_name' in request['args']) »
  @foreach tweet in « result['body'] » identifies TwitterTweets at « request['headers']['Host'] »
    with index « "t" + tweet['id_str'] »
  @ensures « 'count' not in request['args'] or length(result) <= max(200, request['args']['count']) »
  @ensures « for tweet in result['body']: assert parsedate_tz(tweet['created_at']) != None »

  /1.1/statuses/retweet/<id>.json(request)
  @where index is « "t" + request['args']['tweet_id'] »
  @ensures « 'errors' not in result['body'] »
}
```

### B.3 Chess

```
from chess.pgn import read_game

service Chess {
  GetMyGames(username, password)
  @foreach g in « result »  identifies Chess at receiver with index « (g['gameId'], moves(g), False) »
  @foreach g in « result » identifies Chess  at receiver with index « (g['gameId'], moves(g), True) »
  when « g['drawOffered'] == True »
  @ensures «
    for game in result:
      try: read_game(game['moves'])
      except: return False »

  MakeAMove(username, password, gameId, resign, acceptDraw, movecount, myMove,
            offerDraw, claimDraw, myMessage)
  @where index is « (gameId, movecount, acceptDraw) »
  @ensures « result != "NoDrawWasOffered" and result != "InvalidGameID" »
}
```

## C ADDITIONAL CASE STUDY: AIRLINE RESERVATIONS

The airline reservation open-source case study [McGregor et al. 2012; Wilde et al. 2012] models an airline reservation system, and was developed to provide researchers and educators with a simple but complete service-oriented application. With respect to the challenges we discuss in Section 1, it demonstrates that (i) Whip can operate on top of yet another interface abstraction (SOAP with a WSDL interface); and (ii) Whip can express and enforce contracts between components implemented in different languages (Python and PHP for client and server respectively).

The text documentation of the case study informally describes the component interfaces, and the WSDL specification captures some of the interface's syntactic properties. However, WSDL cannot express many of the properties in the text documentation. We wrote Whip contracts for some of these additional properties.

**Well-formed passenger information.** The documentation states that passengers' first and last names cannot contain whitespace. This ensures consistent treatment of passenger names by all services and applies to all services that use passenger data.

With Whip, we are able to ensure that passenger information is well-formed by adding the following tags for the bookSeat operation to the contract of the airline server (We also add similar tags for other uses of passenger information.)

```
bookSeat(bookingRequestNumber, flightId, passenger)
@requires « ' ' not in passenger.firstName »
@requires « ' ' not in passenger.lastName »
```

We tested the effectiveness of this contract by removing the passenger data validity check from the reference implementation of the bookSeat operation. Similar defensive code that checks this property appears in multiple places in the case study's code base. Without Whip and the defensive code, a client is able to book a flight with invalid passenger information. With Whip and the above contract, the violation was reported and the client was blamed. Thus the Whip contract is as effective as defensive code injected in multiple places in the service's code.