Gavial, Programming the web with multi-tier FRP: Semantics

This document contains the denotational semantics of 'Gavial: Programming the web with multi-tier FRP'.

First, we define our semantic domain, secondly we list all core operations regarding clients, events and behaviors. Our project has three kinds of behaviors (plus events) on three tiers, in order to cut down on the size and make this document as legible as possible, our core operations section does not contain the written semantics for all tiers. Instead we limit ourselves to the client tier, the basic operations have no tier-specific abnormalities and the definitions for the session and application tier are nearly identical. For examples of written semantics on core operations on a different tier we have used application tier core operations in the paper.

Thirdly we introduce the conversion primitives. All operations that do not have an impact on time (i.e., no delays) but convert from one tier to another or from one FRP abstraction to another are given.

Next, we define the boundary operations which give precise semantics to server-to-client and client-to-server primitives.

Finally we finish up by proving some useful properties.

Contents

1 Domain

$$Client \subset \mathbb{N} \qquad \texttt{finite}(Client) \qquad [\![Client]\!] = Client$$

$$\begin{aligned} ClientStatus &= \{Connected, Disconnected\} \\ ClientChange &= ClientStatus \times Client \end{aligned}$$

$$\begin{gathered} Time = \mathbb{R}_{\geq 0} \\ \exists\, Time_0, Time_\infty \in Time \\ \forall c \in Client.\, \exists\, Time_{0,c}, Time_{\infty,c} \in Time \end{gathered}$$

$$ServerSlot = Client \uplus \top$$

$$\begin{aligned} & Time^s = Time \times ServerSlot \\ & \forall s_1, s_2 \in ServerSlot.\, s_1 < s_2 \Leftrightarrow (s_1 = \top \wedge s_2 \neq \top) \vee \\ & \qquad\qquad\qquad\qquad\qquad\qquad (s_1 \neq s_2 \neq \top \wedge s_1 < s_2) \\ & \forall t_1, t_2 \in Time.\, \forall s_1, s_2 \in ServerSlot. \\ & \qquad (t_1, s_1) < (t_2, s_2) \Leftrightarrow t_1 < t_2 \vee (t_1 = t_2 \wedge s_1 < s_2) \end{aligned}$$

$$\begin{aligned} delay_{\mathrm{C}\to\mathrm{S}} &: Time \times Client \to Time^s \\ \forall c \in Client.\, \forall t, t' \in Time.\, (t < t') &\implies delay_{\mathrm{C}\to\mathrm{S}}(t, c) < delay_{\mathrm{C}\to\mathrm{S}}(t', c) \\ \forall c, c' \in Client.\, \forall t, t' \in Time.\, delay_{\mathrm{C}\to\mathrm{S}}(t, c) = (t', c') &\implies t \leq t' \wedge c = c' \end{aligned}$$

$$\begin{aligned} delay_{\mathrm{S}\to\mathrm{C}} &: Time^s \times Client \to Time \\ \forall c \in Client.\, \forall s, s' \in Time^s.\, (s < s') &\implies delay_{\mathrm{S}\to\mathrm{C}}(s, c) < delay_{\mathrm{S}\to\mathrm{C}}(s', c) \\ \forall c \in Client.\, \forall (t, slot) \in Time^s.\, delay_{\mathrm{S}\to\mathrm{C}}&((t, slot), c) > t \end{aligned}$$

$$\begin{aligned} Time^s_{0,c} &= delay_{\mathrm{C}\to\mathrm{S}}(Time_{0,c}, c) \\ Time^s_{\infty,c} &= delay_{\mathrm{C}\to\mathrm{S}}(Time_{\infty,c}, c) \end{aligned}$$

$$\begin{aligned} \forall c \in Client.\, (Time_0 < Time_{0,c} < Time_{\infty,c} < Time_\infty) \\ \forall c \in Client.\, (Time^s_0 < Time^s_{0,c} < Time^s_{\infty,c} < Time^s_\infty) \end{aligned}$$

$$[\![Event^C_\tau]\!] = \left\{ \begin{array}{l} e \in Client \to \mathcal{P}(Time \times [\![\tau]\!]) \mid \\ \forall c \in Client.\, \begin{pmatrix} \texttt{finite}(e(c)) \\ \wedge \forall (t, v), (t', v') \in e(c).\, t = t' \Rightarrow v = v' \\ \wedge \forall (t, -) \in e(c).\, Time_{0,c} < t < Time_{\infty,c} \end{pmatrix} \end{array} \right\}$$

$$[\![Event^S_\tau]\!] = \left\{ e \in Client \to \mathcal{P}(Time^s \times [\![\tau]\!]) \;\middle|\; \forall c \in Client.\, \begin{pmatrix} \texttt{finite}(e(c)) \\ \wedge \forall (s, v), (s', v') \in e(c).\, s = s' \Rightarrow v = v' \\ \wedge \forall (s, -) \in e(c).\, Time^s_{0,c} < s < Time^s_{\infty,c} \end{pmatrix} \right\}$$

$$[\![Event^A_\tau]\!] = \left\{\begin{array}{l} e \in \mathcal{P}(Time^s \times [\![\tau]\!]) \mid \\ \texttt{finite}(e) \\ \wedge \forall (s,v),(s',v') \in e.\, s = s' \Rightarrow v = v' \\ \wedge \forall (s,-) \in e.\, Time^s_0 < s < Time^s_\infty \end{array}\right\}$$

$$[\![Behavior^C_\tau]\!] = \left\{\begin{array}{l} b \in Client \to Time \rightharpoonup [\![\tau]\!] \mid \\ \forall c \in Client.\, \texttt{dom}(b(c)) = \\ \quad \{t \in Time \mid Time_{0,c} \le t < Time_{\infty,c}\} \end{array}\right\}$$

$$[\![Behavior^S_\tau]\!] = \left\{b \in Client \to Time^s \rightharpoonup [\![\tau]\!] \mid \forall c \in Client.\, \texttt{dom}(b(c)) = \left\{s \in Time^s \mid Time^s_{0,c} \le s < Time^s_{\infty,c}\right\}\right\}$$

$$[\![Behavior^A_\tau]\!] = \left\{\begin{array}{l} b \in Time^s \rightharpoonup [\![\tau]\!] \mid \\ \texttt{dom}(b) = \left\{\begin{array}{l} s \in Time^s \mid \\ Time^s_0 \le s < Time^s_\infty \end{array}\right\} \end{array}\right\}$$

$$[\![IncBehavior^C_{\tau,\delta}]\!] = \left\{(e,v,f) \in [\![Event^C_\delta]\!] \times (Client \to [\![\tau]\!]) \times Client \to ([\![\tau]\!] \times [\![\delta]\!] \to [\![\tau]\!]) \quad \right\}$$

$$[\![IncBehavior^S_{\tau,\delta}]\!] = \left\{(e,v,f) \in [\![Event^S_\delta]\!] \times (Client \to [\![\tau]\!]) \times (Client \to ([\![\tau]\!] \times [\![\delta]\!] \to [\![\tau]\!])) \quad \right\}$$

$$[\![IncBehavior^A_{\tau,\delta}]\!] = \left\{\begin{array}{l} (e,v_0,f) \in \\ \quad [\![Event^A_\delta]\!] \times [\![\tau]\!] \times ([\![\tau]\!] \times [\![\delta]\!] \to [\![\tau]\!]) \end{array}\right\}$$

$$[\![DiscBehavior^C_\tau]\!] = \left\{\begin{array}{r} (e,v,f) \in [\![IncBehavior^C_{\tau,\tau}]\!] \mid \\ f = \lambda c.\, \lambda v.\, \lambda v'.\, v' \end{array}\right\}$$

$$[\![DiscBehavior^S_\tau]\!] = \left\{\begin{array}{r} (e,v,f) \in [\![IncBehavior^S_{\tau,\tau}]\!] \mid \\ f = \lambda c.\, \lambda v.\, \lambda v'.\, v' \end{array}\right\}$$

$$[\![DiscBehavior^A_\tau]\!] = \left\{\begin{array}{l} (e,v_0,f) \in [\![IncBehavior^A_{\tau,\tau}]\!] \mid \\ f = \lambda v.\, \lambda v'.\, v' \end{array}\right\}$$

2 Core Operations

2.1 Client Information

Definition 1.

$$client : Behavior^S_{Client}$$
$$[\![client]\!] = \lambda c.\, \lambda s. \begin{cases} c & \textbf{if } Time^s_{0,c} \le s < Time^s_{\infty,c} \\ \perp & \textbf{otherwise} \end{cases}$$

Theorem 1. *client returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The domain is correct by definition.

□

Definition 2.

$$\begin{aligned} &\mathit{clientChanges} : \mathit{Event}^{A}_{\mathit{ClientChange}} \\ &[\![\mathit{clientChanges}]\!] = \cup_{c \in \mathit{Client}} \{(\mathit{Time}^{s}_{0,c}, (\mathit{Connected}, c)), (\mathit{Time}^{s}_{\infty,c}, (\mathit{Disconnected}, c))\} \end{aligned}$$

Theorem 2. *$[\![\mathit{clientChanges}]\!]$ returns a valid result.*

Proof. There are 3 properties that the resulting value must comply with in order to be valid.

1. The result is finite since there is a finite number of clients.
2. All values are unique in (server)time since both $\mathit{Time}^{s}_{0,c}$ and $\mathit{Time}^{s}_{\infty,c}$ are indexed by c while $\mathit{Time}^{s}_{0,c} < \mathit{Time}^{s}_{\infty,c}$ and are thus client-specific.
3. All values are within the required bounds since $\mathit{Time}^{s}_{0,c}$ and $\mathit{Time}^{s}_{\infty,c}$ is always between Time^{s}_{0} and $\mathit{Time}^{s}_{\infty}$

□

2.2 Events

Definition 3.

$$\begin{aligned} \mathit{map} &: \mathit{Event}^{C}_{\alpha} \to (\alpha \to \beta) \to \mathit{Event}^{C}_{\beta} \\ [\![\mathit{map}]\!](e, f) &= \lambda c. \{(t, f(v) \mid (t, v) \in e(c)\} \end{aligned}$$

Theorem 3. *If the arguments are valid, $[\![\mathit{map}]\!](e, f)$ returns a valid result.*

Proof. There are 3 properties that the resulting value must comply with in order to be valid.

1. No additional values are added so the result remains finite.
2. No changes are made to the time component of the event values, all relevant properties of e hold.
3. See above.

□

Definition 4.

$$\begin{aligned} \mathit{map} &: \mathit{Event}^{S}_{\alpha} \to (\alpha \to \beta) \to \mathit{Event}^{S}_{\beta} \\ [\![\mathit{map}]\!](e, f) &= \lambda c. \{(s, f(v) \mid (s, v) \in e(c)\} \end{aligned}$$

Theorem 4. *If the arguments are valid, $[\![\mathit{map}]\!](e, f)$ returns a valid result.*

Proof. Similar to Event^{C} □

Definition 5.

$$\begin{aligned} \mathit{map} &: \mathit{Event}^{A}_{\alpha} \to (\alpha \to \beta) \to \mathit{Event}^{A}_{\beta} \\ [\![\mathit{map}]\!](e, f) &= \{(s, f(v) \mid (s, v) \in e\} \end{aligned}$$

Theorem 5. *If the arguments are valid, $[\![\mathit{map}]\!](e, f)$ returns a valid result.*

Proof. Similar to Event^{C} □

Definition 6.

$$\begin{aligned} \mathit{filter} &: \mathit{Event}^{C}_{\tau} \to (\tau \to \mathit{Bool}) \to \mathit{Event}^{C}_{\tau} \\ [\![\mathit{filter}]\!](e, f) &= \lambda c. \{(t, v) \mid (t, v) \in e(c) \wedge f(v)\} \end{aligned}$$

Theorem 6. *If the arguments are valid, $[\![\mathit{filter}]\!](e, f)$ returns a valid result.*

Proof. There are 3 properties that the resulting value must comply with in order to be valid.

1. Event values are only being removed and not added so it remains finite.
2. No changes are made to the time component of the event values, all relevant properties of e hold.

3. See above.

$\square$

Definition 7.

$$\begin{aligned} \mathit{filter} : \mathit{Event}^S_\tau \to (\tau \to \mathit{Bool}) \to \mathit{Event}^S_\tau \\ [\![\mathit{filter}]\!](e, f) = \lambda c. \{(s, v) \mid (s, v) \in e(c) \wedge f(v)\} \end{aligned}$$

Theorem 7. *If the arguments are valid, $[\![\mathit{filter}]\!](e, f)$ returns a valid result.*

Proof. Similar to Event^C $\square$

Definition 8.

$$\begin{aligned} \mathit{filter} : \mathit{Event}^A_\tau \to (\tau \to \mathit{Bool}) \to \mathit{Event}^A_\tau \\ [\![\mathit{filter}]\!](e, f) = \{(s, v) \mid (s, v) \in e \wedge f(v)\} \end{aligned}$$

Theorem 8. *If the arguments are valid, $[\![\mathit{filter}]\!](e, f)$ returns a valid result.*

Proof. Similar to Event^C $\square$

Definition 9.

$$\begin{aligned} \mathit{union} : \mathit{Event}^C_\tau \to \mathit{Event}^C_\tau \to (\tau \to \tau \to \tau) \to \mathit{Event}^C_\tau \\ [\![\mathit{union}]\!](e, e', f) = \lambda c. \, \mathbf{let} \quad \mathit{left} &= \big\{(t, v) \mid (t, v) \in e(c) \wedge \forall (t', -) \in e'(c).\, t \neq t'\big\} \\ \mathit{both} &= \big\{(t, f(v, v')) \mid (t, v) \in e(c) \wedge (t', v') \in e'(c) \wedge t = t'\big\} \\ \mathit{right} &= \big\{(t, v) \mid (t, v) \in e'(c) \wedge \forall (t', -) \in e(c).\, t \neq t'\big\} \\ \mathbf{in}\ \mathit{left} \cup \mathit{both} \cup \mathit{right} \end{aligned}$$

Theorem 9. *If the arguments are valid, $[\![\mathit{union}]\!](e, e', f)$ returns a valid result.*

Proof. There are 3 properties that the resulting value must comply with in order to be valid.

1. The union of three finite sets remains finite.
2. The event values are unique in time by construction, *both* handles exactly this case, merging two values into one with f whenever the event values have the same time component.
3. The event values are within the correct bounds since both e and e' are too.

$\square$

Definition 10.

$$\begin{aligned} \mathit{union} : \mathit{Event}^S_\tau \to \mathit{Event}^S_\tau \to (\tau \to \tau \to \tau) \to \mathit{Event}^S_\tau \\ [\![\mathit{union}]\!](e, e', f) = \lambda c. \, \mathbf{let} \quad \mathit{left} &= \big\{(s, v) \mid (s, v) \in e(c) \wedge \forall (s', -) \in e'(c).\, s \neq s'\big\} \\ \mathit{both} &= \big\{(s, f(v, v')) \mid (s, v) \in e(c) \wedge (s', v') \in e'(c) \wedge s = s'\big\} \\ \mathit{right} &= \big\{(s, v) \mid (s, v) \in e'(c) \wedge \forall (s', -) \in e(c).\, s \neq s'\big\} \\ \mathbf{in}\ \mathit{left} \cup \mathit{both} \cup \mathit{right} \end{aligned}$$

Theorem 10. *If the arguments are valid, $[\![\mathit{union}]\!](e, e', f)$ returns a valid result.*

Proof. Similar to Event^C $\square$

Definition 11.

$$\begin{aligned} & \mathit{union} : \mathit{Event}^A_\tau \to \mathit{Event}^A_\tau \to (\tau \to \tau \to \tau) \to \mathit{Event}^A_\tau \\ & [\![\mathit{union}]\!](e, e', f) = \\ & \mathbf{let} \quad \mathit{left} = \big\{(s, v) \mid (s, v) \in e \wedge \forall (s', -) \in e'.\, s \neq s'\big\} \\ & \qquad \mathit{both} = \big\{(s, f(v, v')) \mid (s, v) \in e \wedge (s', v') \in e' \wedge s = s'\big\} \\ & \qquad \mathit{right} = \big\{(s, v) \mid (s, v) \in e' \wedge \forall (s', -) \in e.\, s \neq s'\big\} \\ & \mathbf{in}\ \mathit{left} \cup \mathit{both} \cup \mathit{right} \end{aligned}$$

Theorem 11. *If the arguments are valid, $[\![\mathit{union}]\!](e, e', f)$ returns a valid result.*

Proof. Similar to Event^C $\square$

2.3 Behavior

Definition 12.

$$\mathit{constant} : \alpha \to \mathit{Behavior}^C_\alpha$$

$$[\![\mathit{constant}]\!](a) = \lambda c.\, \lambda t. \begin{cases} a & \textbf{if } \mathit{Time}_{0,c} \leq t < \mathit{Time}_{\infty,c} \\ \bot & \textbf{otherwise} \end{cases}$$

Theorem 12. *If the arguments are valid,* $[\![\mathit{constant}]\!](a)$ *returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The domain is correct by definition.

□

Definition 13.

$$\mathit{constant} : \alpha \to \mathit{Behavior}^S_\alpha$$

$$[\![\mathit{constant}]\!](a) = \lambda c.\, \lambda s. \begin{cases} a & \textbf{if } \mathit{Time}^s_{0,c} \leq s < \mathit{Time}^s_{\infty,c} \\ \bot & \textbf{otherwise} \end{cases}$$

Theorem 13. *If the arguments are valid,* $[\![\mathit{constant}]\!](a)$ *returns a valid result.*

Proof. Similar to Event^C □

Definition 14.

$$\mathit{constant} : \alpha \to \mathit{Behavior}^A_\alpha$$

$$[\![\mathit{constant}]\!](a) = \lambda s. \begin{cases} a & \textbf{if } \mathit{Time}^s_0 \leq s < \mathit{Time}^s_\infty \\ \bot & \textbf{otherwise} \end{cases}$$

Theorem 14. *If the arguments are valid,* $[\![\mathit{constant}]\!](a)$ *returns a valid result.*

Proof. Similar to Event^C □

Definition 15.

$$\mathit{map2} : \mathit{Behavior}^C_\alpha \to \mathit{Behavior}^C_\beta \to (\alpha \to \beta \to \gamma) \to \mathit{Behavior}^C_\gamma$$

$$[\![\mathit{map2}]\!](b, b', f) = \lambda c.\, \lambda t. \begin{cases} f(b(c)(t), b'(c)(t)) & \textbf{if } \mathit{Time}_{0,c} \leq t < \mathit{Time}_{\infty,c} \\ \bot & \textbf{otherwise} \end{cases}$$

Theorem 15. *If the arguments are valid,* $[\![\mathit{map2}]\!](b, b', f)$ *returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The domain is valid by construction.

□

Definition 16.

$$\mathit{map2} : \mathit{Behavior}^S_\alpha \to \mathit{Behavior}^S_\beta \to (\alpha \to \beta \to \gamma) \to \mathit{Behavior}^S_\gamma$$

$$[\![\mathit{map2}]\!](b, b', f) = \lambda c.\, \lambda s. \begin{cases} f(b(c)(s), b'(c)(s)) & \textbf{if } \mathit{Time}^s_{0,c} \leq s < \mathit{Time}^s_{\infty,c} \\ \bot & \textbf{otherwise} \end{cases}$$

Theorem 16. *If the arguments are valid,* $[\![\mathit{map2}]\!](b, b', f)$ *returns a valid result.*

Proof. Similar to Event^C □

Definition 17.

$$\mathit{map2} : \mathit{Behavior}^A_\alpha \to \mathit{Behavior}^A_\beta \to (\alpha \to \beta \to \gamma) \to \mathit{Behavior}^A_\gamma$$

$$[\![\mathit{map2}]\!](b, b', f) = \lambda s. \begin{cases} f(b(s), b'(s)) & \textbf{if } \mathit{Time}^s_0 \leq s < \mathit{Time}^s_\infty \\ \bot & \textbf{otherwise} \end{cases}$$

Theorem 17. *If the arguments are valid,* $[\![\mathit{map2}]\!](b, b', f)$ *returns a valid result.*

Proof. Similar to Event^C □

2.4 Incremental Behavior

Definition 18.

$$
\begin{aligned}
\mathit{constant} &: \alpha \to \mathit{IncBehavior}^{C}_{\alpha,\delta} \\
[\![\mathit{constant}]\!](a) &= (\lambda c.\, \emptyset, \lambda c.\, a, \lambda c.\, \lambda v.\, \lambda d.\, v)
\end{aligned}
$$

Theorem 18. *If the arguments are valid,* $[\![\mathit{constant}]\!](a)$ *returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The 3 constructed elements of the tuple are of the correct type.

□

Definition 19.

$$
\begin{aligned}
\mathit{constant} &: \alpha \to \mathit{IncBehavior}^{S}_{\alpha,\delta} \\
[\![\mathit{constant}]\!](a) &= (\lambda c.\, \emptyset, \lambda c.\, a, \lambda c.\, \lambda v.\, \lambda d.\, v)
\end{aligned}
$$

Theorem 19. *If the arguments are valid,* $[\![\mathit{constant}]\!](a)$ *returns a valid result.*

Proof. Similar to Event^{C} □

Definition 20.

$$
\begin{aligned}
\mathit{constant} &: \alpha \to \mathit{IncBehavior}^{A}_{\alpha,\delta} \\
[\![\mathit{constant}]\!](a) &= (\emptyset, a, \lambda v.\, \lambda d.\, v)
\end{aligned}
$$

Theorem 20. *If the arguments are valid,* $[\![\mathit{constant}]\!](a)$ *returns a valid result.*

Proof. Similar to Event^{C} □

Definition 21.

$$
\begin{aligned}
&[\![\mathit{Inc}_{\delta,\delta'}]\!] = \{\mathit{Left}(x) \mid x \in [\![\delta]\!]\} \cup \{\mathit{Right}(y) \mid y \in [\![\delta']\!]\} \cup \{\mathit{All}(x,y) \mid x \in [\![\delta]\!], y \in [\![\delta']\!]\} \\
&\mathit{incMap2} : \mathit{IncBehavior}^{C}_{\tau,\delta} \to \mathit{IncBehavior}^{C}_{\tau',\delta'} \to (\tau \to \tau' \to \tau'') \\
&\qquad\qquad \to (\tau \to \tau' \to \mathit{Inc}_{\delta,\delta'} \rightharpoonup \delta'') \to (\tau'' \to \delta'' \to \tau'') \to \mathit{IncBehavior}^{C}_{\tau'',\delta''}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathit{incMap2}]\!]((e,v,f),(e',v',f'),\mathit{fi},\mathit{fd},\mathit{ff}) = \\
&\qquad \textbf{let} \quad b = [\![\mathit{convert}]\!]((e,v,f)) \\
&\qquad\qquad b' = [\![\mathit{convert}]\!]((e',v',f')) \\
&\qquad\qquad bb = [\![\mathit{map2}]\!](b, b', \lambda x.\, \lambda y.\, (x,y))) \\
&\qquad\qquad de = [\![\mathit{map}]\!](e, \lambda x.\, \mathit{Left}(x)) \\
&\qquad\qquad de' = [\![\mathit{map}]\!](e', \lambda x.\, \mathit{Right}(x)) \\
&\qquad\qquad de'' = [\![\mathit{union}]\!](de, de', \lambda(\mathit{Left}(x)).\, \lambda(\mathit{Right}(y)).\, \mathit{All}(x,y) \\
&\qquad\qquad e'' = [\![\mathit{snapshot}]\!](bb, [\![\mathit{map}]\!](de'', \lambda dev.\, \lambda(bv, bv').\, \mathit{fd}(bv)(bv')(dev))) \\
&\qquad \textbf{in}([\![\mathit{filter}]\!](e'', \lambda x.\, x \neq \bot), \lambda c.\, \mathit{fi}(v(c), v'(c)), \mathit{ff})
\end{aligned}
$$

Theorem 21. *If the arguments are valid,* $[\![\mathit{incMap2}]\!]((e,v,f),(e',v',f'),\mathit{fi},\mathit{fd},\mathit{ff})$ *returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The 3 constructed elements of the tuple are of the correct type.

□

Definition 22.

$$
\begin{aligned}
\mathit{incMap2} : \mathit{IncBehavior}^S_{\tau,\delta} \to \mathit{IncBehavior}^S_{\tau',\delta'} &\to (\tau \to \tau' \to \tau'') \\
&\to (\tau \to \tau' \to \mathit{Inc}_{\delta,\delta'} \rightharpoonup \delta'') \to (\tau'' \to \delta'' \to \tau'') \to \mathit{IncBehavior}^S_{\tau'',\delta''}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathit{incMap2}]\!]((e,v,f),(e',v',f'),\mathit{fi},\mathit{fd},\mathit{ff}) = \\
&\qquad \mathbf{let}\quad b = [\![\mathit{convert}]\!]((e,v,f)) \\
&\qquad\qquad b' = [\![\mathit{convert}]\!]((e',v',f')) \\
&\qquad\qquad bb = [\![\mathit{map2}]\!](b, b', \lambda x.\, \lambda y.\, (x,y))) \\
&\qquad\qquad de = [\![\mathit{map}]\!](e, \lambda x.\, \mathit{Left}(x)) \\
&\qquad\qquad de' = [\![\mathit{map}]\!](e', \lambda x.\, \mathit{Right}(x)) \\
&\qquad\qquad de'' = [\![\mathit{union}]\!](de, de', \lambda(\mathit{Left}(x)).\, \lambda(\mathit{Right}(y)).\, \mathit{All}(x,y) \\
&\qquad\qquad e'' = [\![\mathit{snapshot}]\!](bb, [\![\mathit{map}]\!](de'', \lambda dev.\, \lambda(bv,bv').\, \mathit{fd}(bv)(bv')(dev))) \\
&\qquad \mathbf{in}([\![\mathit{filter}]\!](e'', \lambda x.\, x \neq \bot), \lambda c.\, \mathit{fi}(v(c), v'(c)), \mathit{ff})
\end{aligned}
$$

Theorem 22. *If the arguments are valid, $[\![\mathit{incMap2}]\!]((e,v,f),(e',v',f'),\mathit{fi},\mathit{fd},\mathit{ff})$ returns a valid result.*

Proof. Similar to Event^C □

Definition 23.

$$
\begin{aligned}
\mathit{incMap2} : \mathit{IncBehavior}^A_{\tau,\delta} \to \mathit{IncBehavior}^A_{\tau',\delta'} &\to (\tau \to \tau' \to \tau'') \\
&\to (\tau \to \tau' \to \mathit{Inc}_{\delta,\delta'} \rightharpoonup \delta'') \to (\tau'' \to \delta'' \to \tau'') \to \mathit{IncBehavior}^A_{\tau'',\delta''}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathit{incMap2}]\!]((e,v,f),(e',v',f'),\mathit{fi},\mathit{fd},\mathit{ff}) = \\
&\qquad \mathbf{let}\quad b = [\![\mathit{convert}]\!]((e,v,f)) \\
&\qquad\qquad b' = [\![\mathit{convert}]\!]((e',v',f')) \\
&\qquad\qquad bb = [\![\mathit{map2}]\!](b, b', \lambda x.\, \lambda y.\, (x,y))) \\
&\qquad\qquad de = [\![\mathit{map}]\!](e, \lambda x.\, \mathit{Left}(x)) \\
&\qquad\qquad de' = [\![\mathit{map}]\!](e', \lambda x.\, \mathit{Right}(x)) \\
&\qquad\qquad de'' = [\![\mathit{union}]\!](de, de', \lambda(\mathit{Left}(x)).\, \lambda(\mathit{Right}(y)).\, \mathit{All}(x,y) \\
&\qquad\qquad e'' = [\![\mathit{snapshot}]\!](bb, [\![\mathit{map}]\!](de'', \lambda dev.\, \lambda(bv,bv').\, \mathit{fd}(bv)(bv')(dev))) \\
&\qquad \mathbf{in}([\![\mathit{filter}]\!](e'', \lambda x.\, x \neq \bot), \mathit{fi}(v, v'), \mathit{ff})
\end{aligned}
$$

Theorem 23. *If the arguments are valid, $[\![\mathit{incMap2}]\!]((e,v,f),(e',v',f'),\mathit{fi},\mathit{fd},\mathit{ff})$ returns a valid result.*

Proof. Similar to Event^C □

2.5 Discrete Behavior

$$
\mathit{discMap2} : \mathit{DiscBehavior}^C_\alpha \to \mathit{DiscBehavior}^C_\beta \to (\alpha \to \beta \to \tau) \to \mathit{DiscBehavior}^C_\tau
$$

$$
[\![\mathit{discMap2}]\!](b, b', f) = [\![\mathit{incMap2}]\!]\left(b, b', f, \lambda x.\, \lambda y.\, \lambda d.\, \begin{cases} f(x',y) & \mathbf{if}\ d = \mathit{Left}(x') \\ f(x,y') & \mathbf{if}\ d = \mathit{Right}(y') \\ f(x',y') & \mathbf{if}\ d = \mathit{All}(x',y') \end{cases}, \lambda c.\, \lambda v.\, \lambda v'.\, v' \right)
$$

$$
\mathit{discMap2} : \mathit{DiscBehavior}^S_\alpha \to \mathit{DiscBehavior}^S_\beta \to (\alpha \to \beta \to \tau) \to \mathit{DiscBehavior}^S_\tau
$$

$$
[\![\mathit{discMap2}]\!](b, b', f) = [\![\mathit{incMap2}]\!]\left(b, b', f, \lambda x.\, \lambda y.\, \lambda d.\, \begin{cases} f(x',y) & \mathbf{if}\ d = \mathit{Left}(x') \\ f(x,y') & \mathbf{if}\ d = \mathit{Right}(y') \\ f(x',y') & \mathbf{if}\ d = \mathit{All}(x',y') \end{cases}, \lambda c.\, \lambda v.\, \lambda v'.\, v' \right)
$$

$$discMap2 : DiscBehavior^A_\alpha \to DiscBehavior^A_\beta \to (\alpha \to \beta \to \tau) \to DiscBehavior^A_\tau$$

$$[\![discMap2]\!](b, b', f) = [\![incMap2]\!]\left(b, b', f, \lambda x.\, \lambda y.\, \lambda d. \begin{cases} f(x', y) & \textbf{if } d = Left(x') \\ f(x, y') & \textbf{if } d = Right(y') \\ f(x', y') & \textbf{if } d = All(x', y') \end{cases}, \lambda v.\, \lambda v'.\, v'\right)$$

We can define discrete map two in terms of the incremental version since we defined earlier that semantically discrete behaviors are a subset of incremental behaviors.

3 Conversions

3.1 Events

Definition 24.

$$toApplication : Event^S_\tau \to Event^A_{Client \rightharpoonup \tau}$$

$$[\![toApplication]\!](e) = \textbf{let } allOccurrences = \cup_{c \in Client}\{(s, (c, v)) \mid (s, v) \in e(c)\}$$

$$\textbf{in} \cup_{s \in Time^s.\, Time^s_0 < s < Time^s_\infty} \left\{(s, set) \;\middle|\; \begin{array}{r} set' = \{(c, v) \mid (s', (c, v)) \in allOccurrences \;\; \wedge s = s'\} \wedge \\ set' \neq \emptyset \wedge set = \textbf{fromSet}(set') \end{array}\right\}$$

$$\textbf{fromSet} : \mathcal{P}(Client \times \tau) \to Client \rightharpoonup \tau$$

$$\textbf{fromSet}(s) = \lambda c. \begin{cases} v & \textbf{if } (v, c) \in s \\ \bot & \textbf{otherwise} \end{cases}$$

Theorem 24. *If the arguments are valid, $[\![toApplication]\!](e)$ returns a valid result.*

Proof. There are 3 properties that the resulting value must comply with in order to be valid.

1. There is a finite number of clients which implies that a union of all values for all connected clients is finite (a union of finite amount of finite sets).

2. We group all occurrences per time slot in a partial function using **fromSet**, as such the resulting set is unique in time since the original occurrences were unique in time per client.

3. $Event^S$s have time bounds that are always within the bounds of $Event^A$s and we only take sets that contain values from $Event^S$s.

□

Definition 25.

$$toSession : Event^A_\tau \to Event^S_\tau$$

$$[\![toSession]\!](e) = \lambda c.\, \{(s, v) \mid (s, v) \in e \wedge Time^s_{0,c} < s < Time^s_{\infty,c}\}$$

Theorem 25. *If the arguments are valid, $[\![toSession]\!](e)$ returns a valid result.*

Proof. There are 3 properties that the resulting value must comply with in order to be valid.

1. The set of all events for a client is at most the same size and distribution as the set of all events for all clients. The latter is already required to be finite by $Event^A$s.

2. All event values are unique in time for $Event^A$s and per event value only one value is calculated per client so this is obvious.

3. We only select occurrences that fit the required time bounds.

□

3.2 Behavior

Definition 26.

$$toApplication : Behavior^S_\tau \to Behavior^A_{Client \rightharpoonup \tau}$$

$$[\![toApplication]\!](b) = \lambda s. \begin{cases} \lambda c \to b(c)(s) & \textbf{if } Time^s_0 \leq s < Time^s_\infty \\ \bot & \textbf{otherwise} \end{cases}$$

Theorem 26. *If the arguments are valid, $[\![toApplication]\!](b)$ returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The domain of the resulting function is within the required bounds by construction.

□

Definition 27.

$$toSession : Behavior^A_\tau \to Behavior^S_\tau$$

$$[\![toSession]\!](b) = \lambda c. \lambda s. \begin{cases} b(s) & \textbf{if } Time^s_{0,c} \leq s < Time^s_{\infty,c} \\ \bot & \textbf{otherwise} \end{cases}$$

Theorem 27. *If the arguments are valid, $[\![toSession]\!](b)$ returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The domain of the resulting function is within the required bounds by construction.

□

3.3 Incremental Behavior

Definition 28.

$$toApplication : IncBehavior^S_{\tau,\delta} \to IncBehavior^A_{Client \rightharpoonup \tau, Client \rightharpoonup \delta \times (ClientChange \uplus \top)}$$

$$\begin{aligned} [\![toApplication]\!](\ (e, v_0, f)\ ) = \textbf{let} \quad & e' = [\![map]\!]([\![toApplication]\!](e), \lambda map. (map, \top)) \\ & clientChanges' = [\![map]\!]([\![clientChanges]\!], \lambda c. (\lambda\_ . \bot, c)) \\ & u = [\![union]\!](e', clientChanges', \lambda map. \lambda c. (map_1, c_2)) \\ & f' = \lambda v. \lambda d. \lambda c. \begin{cases} v_0(c) & \textbf{if } d.2 = (Connected, c) \\ \bot & \textbf{if } d.2 = (Disconnected, c) \\ v(c) & \textbf{if } d.1(c) = \bot \\ f(c)(v(c))(d.1(c)) & \textbf{otherwise} \end{cases} \\ \textbf{in} & (u, \lambda c \to \bot, f') \end{aligned}$$

Theorem 28. *If the arguments are valid, $[\![toApplication]\!](b)$ returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The 3 constructed elements of the resulting tuple are of the correct type.

□

Definition 29.

$$toSession : IncBehavior^A_{\tau,\delta} \to IncBehavior^S_{\tau,\delta}$$

$$[\![toSession]\!]((e, v, f)) = \big([\![toSession]\!](e), \lambda c. [\![convert]\!]((e, v, f))(Time^s_{0,c}), \lambda c. f\big)$$

Theorem 29. *If the arguments are valid, $[\![toSession]\!]((e, v, f))$ returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The 3 constructed elements of the resulting tuple are of the correct type.

□

3.4 Events ↔ Behaviors

Definition 30.

$$\begin{aligned} &\mathit{snapshot} : \mathit{Event}^C_{\alpha\to\beta} \to \mathit{Behavior}^C_\alpha \to \mathit{Event}^C_\beta \\ &[\![\mathit{snapshot}]\!](e,b) = \lambda c.\,\{(t, v(b(c)(t))) \mid (t,v) \in e(c)\} \end{aligned}$$

Theorem 30. *If the arguments are valid,* $[\![\mathit{snapshot}]\!](e,b)$ *returns a valid result.*

Proof. There are 3 properties that the resulting value must comply with in order to be valid.

1. A value is created for each value in e which is a finite amount.
2. Only one value is created for every value in e so the result remains unique in time. Note that the time bounds on e ensure that $b(c)(t)$ is well defined.
3. The bounds do not change compared to e.

□

Definition 31.

$$\begin{aligned} &\mathit{snapshot} : \mathit{Event}^S_{\alpha\to\beta} \to \mathit{Behavior}^S_\alpha \to \mathit{Event}^S_\beta \\ &[\![\mathit{snapshot}]\!](e,b) = \lambda c.\,\{(s, v(b(c)(s))) \mid (s,v) \in e(c)\} \end{aligned}$$

Theorem 31. *If the arguments are valid,* $[\![\mathit{snapshot}]\!](e,b)$ *returns a valid result.*

Proof. Similar to Event^C □

Definition 32.

$$\begin{aligned} &\mathit{snapshot} : \mathit{Event}^A_{\alpha\to\beta} \to \mathit{Behavior}^A_\alpha \to \mathit{Event}^A_\beta \\ &[\![\mathit{snapshot}]\!](e,b) = \{(s, v(b(s))) \mid (s,v) \in e\} \end{aligned}$$

Theorem 32. *If the arguments are valid,* $[\![\mathit{snapshot}]\!](e,b)$ *returns a valid result.*

Proof. Similar to Event^C □

Definition 33.

$$\begin{aligned} &\mathit{foldP} : \mathit{Event}^C_\delta \to \tau \to (\tau \to \delta \to \tau) \to \mathit{IncBehavior}^C_{\tau,\delta} \\ &[\![\mathit{foldP}]\!](e,v,f) = (e, \lambda c.\, v, \lambda c.\, f) \end{aligned}$$

Theorem 33. *If the arguments are valid,* $[\![\mathit{foldP}]\!](e,v,f)$ *returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The 3 constructed elements of the resulting tuple are of the correct type.

□

Definition 34.

$$\begin{aligned} &\mathit{foldP} : \mathit{Event}^S_\delta \to \tau \to (\tau \to \delta \to \tau) \to \mathit{IncBehavior}^C_{\tau,\delta} \\ &[\![\mathit{foldP}]\!](e,v,f) = (e, \lambda c.\, v, \lambda c.\, f) \end{aligned}$$

Theorem 34. *If the arguments are valid,* $[\![\mathit{foldP}]\!](e,v,f)$ *returns a valid result.*

Proof. Similar to Event^C □

Definition 35.

$$\begin{aligned} &\mathit{foldP} : \mathit{Event}^A_\delta \to \tau \to (\tau \to \delta \to \tau) \to \mathit{IncBehavior}^A_{\tau,\delta} \\ &[\![\mathit{foldP}]\!](e,v,f) = (e, v, f) \end{aligned}$$

Theorem 35. *If the arguments are valid,* $[\![\mathit{foldP}]\!](e,v,f)$ *returns a valid result.*

Proof. Similar to Event^C □

3.5 Incremental Behavior $\to$ Behavior

Definition 36.

$$
\begin{aligned}
&\mathit{convert} : \mathit{IncBehavior}^{C}_{\tau,\delta} \to \mathit{Behavior}^{C}_{\tau} \\
&[\![\mathit{convert}]\!](\ (e, v, f)\ ) = \\
&\lambda c.\, \lambda s. \begin{cases} f(c)(f(c)(...(f(c)(v(c), d_1), ...), d_{n-1}) d_n) \\ \qquad \textbf{if } \mathit{Time}_{0,c} \leq t < \mathit{Time}_{\infty,c} \wedge \\ \qquad\quad e(c) = \{(t_1, d_1), ..., (t_n, d_n), (t_{n+1}, d_{n+1}), ...\} \wedge \\ \qquad\quad t_1 < ... < t_n \leq t < t_{n+1} < ... \\ \perp \quad \textbf{otherwise} \end{cases}
\end{aligned}
$$

Theorem 36. *If the arguments are valid,* $[\![\mathit{convert}]\!](\ (e, v, f)\ )$ *returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The domain of the resulting function is within the required bounds by construction and it is well defined since an order can be given to the timestamps of $e(c)$ due to time being ordered and the amount of event values being finite.

□

Definition 37.

$$
\begin{aligned}
&\mathit{convert} : \mathit{IncBehavior}^{S}_{\tau,\delta} \to \mathit{Behavior}^{S}_{\tau} \\
&[\![\mathit{convert}]\!](\ (e, v, f)\ ) = \\
&\lambda c.\, \lambda s. \begin{cases} f(c)(f(c)(...(f(c)(v(c), d_1), ...), d_{n-1}) d_n) \\ \qquad \textbf{if } \mathit{Time}^{s}_{0,c} \leq s < \mathit{Time}^{s}_{\infty,c} \wedge \\ \qquad\quad e(c) = \{(s_1, d_1), ..., (s_n, d_n), (s_{n+1}, d_{n+1}), ...\} \wedge \\ \qquad\quad s_1 < ... < s_n \leq s < s_{n+1} < ... \\ \perp \quad \textbf{otherwise} \end{cases}
\end{aligned}
$$

Theorem 37. *If the arguments are valid,* $[\![\mathit{convert}]\!](\ (e, v, f)\ )$ *returns a valid result.*

Proof. Similar to Event^{C} □

Definition 38.

$$
\begin{aligned}
&\mathit{convert} : \mathit{IncBehavior}^{A}_{\tau,\delta} \to \mathit{Behavior}^{A}_{\tau} \\
&[\![\mathit{convert}]\!](\ (e, v, f)\ ) = \\
&\lambda s. \begin{cases} f(f(...(f(v, d_1), ...), d_{n-1}) d_n) \\ \qquad \textbf{if } \mathit{Time}^{s}_{0} \leq s < \mathit{Time}^{s}_{\infty} \wedge \\ \qquad\quad e = \{(s_1, d_1), ..., (s_n, d_n), (s_{n+1}, d_{n+1}), ...\} \wedge \\ \qquad\quad s_1 < ... < s_n \leq s < s_{n+1} < ... \\ \perp \quad \textbf{otherwise} \end{cases}
\end{aligned}
$$

Theorem 38. *If the arguments are valid,* $[\![\mathit{convert}]\!](\ (e, v, f)\ )$ *returns a valid result.*

Proof. Similar to Event^{C} □

4 Boundary Operations

4.1 Events

Definition 39.

$$
\begin{aligned}
\mathit{toServer} : \mathit{Event}^{C}_{\tau} \to \mathit{Event}^{S}_{\tau} \\
[\![\mathit{toServer}]\!](e) = \lambda c.\, \{(\mathit{delay}_{\mathrm{C}\to\mathrm{S}}(t, c), v) \mid (t, v) \in e(c)\}
\end{aligned}
$$

Theorem 39. *If the arguments are valid,* $\mathit{toServer}(e)$ *returns a valid result.*

Proof. There are 3 properties that the resulting value must comply with in order to be valid.

1. $Event^C$ is already finite, we do not add any elements.

2. The $delay_{\mathrm{C}\to\mathrm{S}}$ function is injective since it is a strictly monotone function. Since $Event^C$s are unique in time, the resulting $Event^S$ will remain unique in time after applying $delay_{\mathrm{C}\to\mathrm{S}}$.

3. Client events are bound for each client by $Time_{0,c}$ and $Time_{\infty,c}$, since all events are delayed equally per client with $delay_{\mathrm{C}\to\mathrm{S}}$they are bounded by $Time^s_{0,c}$ and $Time^s_{\infty,c}$.

□

Definition 40.

$$
\begin{aligned}
toClient : Event^S_\tau &\to Event^C_\tau \\
[\![toClient]\!](e) &= \lambda c. \left\{ (t, v) \;\middle|\; \begin{array}{l} (s, v) \in e(c) \\ \wedge\, t = delay_{\mathrm{S}\to\mathrm{C}}(s, c) \\ \wedge\, t < Time_{\infty,c} \end{array} \right\}
\end{aligned}
$$

Theorem 40. *If the arguments are valid, $[\![toClient]\!](e)$ returns a valid result.*

Proof. There are 3 properties that the resulting value must comply with in order to be valid.

1. $Event^S$ is already finite, we do not add any elements.

2. The $delay_{\mathrm{S}\to\mathrm{C}}$ function is injective since it is a strictly monotone function. Since $Event^C$s are unique in time, the resulting $Event^S$ will remain unique in time after applying $delay_{\mathrm{S}\to\mathrm{C}}$.

3. Events in $Event^S$s are bounded by $Time^s_{0,c}$ and $Time^s_{\infty,c}$. $delay_{\mathrm{S}\to\mathrm{C}}(Time^s_{0,c}, c)$ is by definition larger than $Time_{0,c}$ (the required lower bound) because delays are increasing.

   The upper bound holds by definition of *toClient*.

□

4.2 Incremental Behavior

Definition 41.

$$
\begin{aligned}
toServer : IncBehavior^C_{\tau,\delta} &\to IncBehavior^S_{\tau,\delta} \\
[\![toServer]\!](b) &= \mathbf{let}(e, v, f) = b \\
&\quad\; \mathbf{in}\,([\![toServer]\!](e), v, f)
\end{aligned}
$$

Theorem 41. *If the arguments are valid, $[\![toServer]\!](b)$ returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The 3 constructed elements of the resulting tuple are of the correct type.

□

Definition 42.

$$
\begin{aligned}
toClient : IncBehavior^S_{\tau,\delta} &\to IncBehavior^C_{\tau,\delta} \\
[\![toClient]\!](i) &= \mathbf{let}(e, v, f) = i \\
&\quad\; \mathbf{in}\,([\![toClient]\!](e), v, f)
\end{aligned}
$$

Theorem 42. *If the arguments are valid, $[\![toClient]\!](i)$ returns a valid result.*

Proof. There is one property that the resulting value must comply with in order to be valid.

1. The 3 constructed elements of the resulting tuple are of the correct type.

□

5 Commutative Replication

Definition 43.

$$[\![toServer]\!]([\![map]\!](e, f)) = [\![map]\!]([\![toServer]\!](e), f)$$

Proof.

$$\begin{aligned}
[\![toServer]\!](\lambda c.\, \{(t, f(v) \mid (t, v) \in e(c)\}) = && [\![map]\!] \\
\lambda c.\, \{(delay_{\mathrm{C}\to\mathrm{S}}(t, c), v) \mid (t, v) \in (\lambda c.\, \{(t, f(v) \mid (t, v) \in e(c)\})(c)\} = && [\![toServer]\!] \\
\lambda c.\, \{(t, f(v), v) \mid (t, v) \in (\lambda c.\, \{(delay_{\mathrm{C}\to\mathrm{S}}(t, c) \mid (t, v) \in e(c)\})(c)\} = && \text{rewrite} \\
\lambda c.\, \{(t, f(v), v) \mid (t, v) \in ([\![toServer]\!](e))(c)\} = && [\![toServer]\!]^{-1} \\
[\![map]\!]([\![toServer]\!](e), f) = && [\![map]\!]^{-1}
\end{aligned}$$

□

Definition 44.

$$[\![toServer]\!]([\![union]\!](e, e', f)) = [\![union]\!]([\![toServer]\!](e), [\![toServer]\!](e'), f)$$

Proof.

$$\begin{aligned}
\lambda c.\,\mathbf{let}\quad & \mathit{left} = \left\{\begin{array}{l}(t, v) \mid \\ (t, v) \in e(c) \\ \wedge \forall (t', -) \in e'(c).\, t \neq t'\end{array}\right\} = && [\![union]\!] \\
& \mathit{both} = \left\{\begin{array}{l}(t, f(v, v')) \mid \\ (t, v) \in e(c) \\ \wedge (t', v') \in e'(c) \\ \wedge t = t'\end{array}\right\} \\
& \mathit{right} = \left\{\begin{array}{l}(t, v) \mid \\ (t, v) \in e'(c) \\ \wedge \forall (t', -) \in e(c).\, t \neq t'\end{array}\right\} \\
& \mathbf{in} [\![toServer]\!](\mathit{left} \cup \mathit{both} \cup \mathit{right}) \\
\lambda c.\,\mathbf{let}\quad & \mathit{left} = \left\{\begin{array}{l}(\mathit{delay}_{\mathrm{C}\to\mathrm{S}}(t, c), v) \mid \\ (t, v) \in e(c) \\ \wedge \forall (t', -) \in e'(c).\, t \neq t'\end{array}\right\} = && [\![toServer]\!] \\
& \mathit{both} = \left\{\begin{array}{l}(\mathit{delay}_{\mathrm{C}\to\mathrm{S}}(t, c), f(v, v')) \mid \\ (t, v) \in e(c) \\ \wedge (t', v') \in e'(c) \\ \wedge t = t'\end{array}\right\} \\
& \mathit{right} = \left\{\begin{array}{l}(\mathit{delay}_{\mathrm{C}\to\mathrm{S}}(t, c), v) \mid \\ (t, v) \in e'(c) \\ \wedge \forall (t', -) \in e(c).\, t \neq t'\end{array}\right\} \\
& \mathbf{in}\ \mathit{left} \cup \mathit{both} \cup \mathit{right} \\
\lambda c.\,\mathbf{let}\quad & eS = \lambda c.\, \{(\mathit{delay}_{\mathrm{C}\to\mathrm{S}}(t, c), v) \mid (t, v) \in e(c)\} = && \text{rewrite} \\
& eS' = \lambda c.\, \{(\mathit{delay}_{\mathrm{C}\to\mathrm{S}}(t, c), v) \mid (t, v) \in e'(c)\} \\
& \mathit{left} = \left\{\begin{array}{l}(t, v) \mid \\ (t, v) \in eS(c) \\ \wedge \forall (t', -) \in eS'(c).\, t \neq t'\end{array}\right\} \\
& \mathit{both} = \left\{\begin{array}{l}(t, f(v, v')) \mid \\ (t, v) \in eS(c) \\ \wedge (t', v') \in eS'(c) \\ \wedge t = t'\end{array}\right\} \\
& \mathit{right} = \left\{\begin{array}{l}(t, v) \mid \\ (t, v) \in eS'(c) \\ \wedge \forall (t', -) \in eS(c).\, t \neq t'\end{array}\right\} \\
& \mathbf{in}\ \mathit{left} \cup \mathit{both} \cup \mathit{right} \\
& [\![union]\!]\left(\begin{array}{l}\lambda c.\, \{(\mathit{delay}_{\mathrm{C}\to\mathrm{S}}(t, c), v) \mid (t, v) \in e(c)\}, \\ \lambda c.\, \{(\mathit{delay}_{\mathrm{C}\to\mathrm{S}}(t, c), v) \mid (t, v) \in e'(c)\}, \\ f\end{array}\right) = && [\![union]\!]^{-1} \\
& [\![union]\!]([\![toServer]\!](e), [\![toServer]\!](e'), f) = && [\![toServer]\!]
\end{aligned}$$

□