(Sketches of) Proofs for Modular Analysis

Joonhyup Lee

September 13, 2023

1 Preliminaries

In this section, we will define:

1. The *abstract syntax* of the language under discussion.
2. The *semantic domains* that are used to define the semantics of the language.
3. The concrete and abstract versions of the *operational semantics* of the language.
4. The *collecting semantics* and the *abstract semantics* of the language.

and sketch the proofs for:

1. The well-definedness of the operational semantics.
2. The Galois connection between the collecting and abstract semantics.
3. The computability of the abstract semantics.

1.1 Definitions

1.1.1 Abstract Syntax

The language is basically an extension of untyped lambda calculus with modules and the linking construct. $e_1 \rtimes e_2$ means that e_1 is a module that is evaluated first to a *context*, and that e_2 is evaluated under the exported context.

Identifiers	x, M	$\in$	Var	
Expression	e	$\to$	x	value identifier
		$\mid$	$\lambda x.e$	function
		$\mid$	$e\ e$	application
		$\mid$	$e \rtimes e$	linked expression
		$\mid$	ε	empty module
		$\mid$	M	module identifier
		$\mid$	`let` $x\ e\ e$	binding expression
		$\mid$	`let` $M\ e\ e$	binding module

Figure 1: Abstract syntax of the simple module language.

1.1.2 Semantic Domains

As it always must be, the domains used in the concrete semantics and the abstract semantics must be connected in a "coherent" way. Our definition of semantics is parametrized by a quadruple $(\mathbb{T}, \leq, \widehat{\mathbb{T}}, \hat{\alpha})$, where:

1. $\leq$ is a total order on $\mathbb{T}$.
2. $\widehat{\mathbb{T}}$ is a (nonempty) finite set.
3. $\hat{\alpha} \in \mathbb{T} \to \widehat{\mathbb{T}}$, and there exists a $\hat{\alpha}^{-1} \in (\mathbb{T} \times \widehat{\mathbb{T}}) \to \mathbb{T}$ that satisfies:

$$\forall t \in \mathbb{T} : t < \hat{\alpha}^{-1}(t, \_) \qquad \text{and} \qquad \forall \hat{t} \in \widehat{\mathbb{T}} : \hat{t} = \hat{\alpha}(\hat{\alpha}^{-1}(\_, \hat{t}))$$

$$
\begin{array}{rlll}
\text{Abstract Time} & \hat{t} & \in & \widehat{\mathbb{T}} \\
\text{Environment/Context} & \widehat{C} & \in & \widehat{\text{Ctx}} \\
\text{Value of expressions} & \hat{v} & \in & \widehat{\text{Val}} \subseteq \text{Expr} \times \widehat{\text{Ctx}} \\
\text{Value of expressions/modules} & \widehat{V} & \in & \widehat{\text{ValCtx}} \triangleq \widehat{\text{Val}} \uplus \widehat{\text{Ctx}} \\
\text{Abstract Memory} & \widehat{m} & \in & \widehat{\text{Mem}} \triangleq \widehat{\mathbb{T}} \xrightarrow{\text{fin}} \wp(\widehat{\text{Val}}) \\
\text{Abstract State} & \hat{s} & \in & \widehat{\text{State}} \triangleq \widehat{\text{Ctx}} \times \widehat{\text{Mem}} \times \widehat{\mathbb{T}} \\
\text{Abstract Result} & \hat{r} & \in & \widehat{\text{Result}} \triangleq \widehat{\text{ValCtx}} \times \widehat{\text{Mem}} \times \widehat{\mathbb{T}} \\
\text{Abstract Tick} & \widehat{\text{tick}} & \in & \widehat{\text{Tick}} \triangleq \widehat{\text{State}} \times \text{Var} \times \widehat{\text{Val}} \to \widehat{\mathbb{T}} \\
\text{Context} & \widehat{C} & \to & [] \qquad \text{empty stack} \\
 & & | & (x, \hat{t}) :: \widehat{C} \qquad \text{expression binding} \\
 & & | & (M, \widehat{C}) :: \widehat{C} \qquad \text{module binding} \\
\text{Value of expressions} & \hat{v} & \to & \langle \lambda x.e, \widehat{C} \rangle \qquad \text{closure}
\end{array}
$$

Figure 2: Definition of the semantic domains in the abstract case. By $\wp(S)$ we mean the powerset of S.

As will be elaborated, these conditions ensure that the concrete and the abstract semantics are soundly connected. Now we give the semantic domains that are parametrized by the choice of $(\mathbb{T}, \leq, \widehat{\mathbb{T}}, \hat{\alpha})$.

We first introduce sets that are used to describe the abstract version of the operational semantics. Among these sets, the most important component for the analysis designer is the $\widehat{\text{tick}}$ component. The $\widehat{\text{tick}}$ function is what determines the granularity of the analysis. Since it takes in $(\hat{s}, x, \hat{v})$ and returns a new $\hat{t}$, the timestamps can encode a variety of context information. Increasing the variety of information encoded in $\hat{t}$ increases the accuracy of the analysis.

Naturally, all $\widehat{\text{tick}}$ functions must have a concrete tick that mirrors its action. Since the abstract time domain $\widehat{\mathbb{T}}$ and the translation function $\hat{\alpha}$ meets the basic list of requirements, for each $\widehat{\text{tick}}$ there exists a tick that *respects* $\widehat{\text{tick}}$ and produces *fresh* timestamps. To elaborate, we must take a look at the definitions for the concrete semantic domains.

$$
\begin{array}{rlll}
\text{Time} & t & \in & \mathbb{T} \\
\text{Environment/Context} & C & \in & \text{Ctx} \\
\text{Value of expressions} & v & \in & \text{Val} \subseteq \text{Expr} \times \text{Ctx} \\
\text{Value of expressions/modules} & V & \in & \text{ValCtx} \triangleq \text{Val} \uplus \text{Ctx} \\
\text{Memory} & m & \in & \text{Mem} \triangleq \mathbb{T} \xrightarrow{\text{fin}} \text{Val} \\
\text{State} & s & \in & \text{State} \subseteq \text{Ctx} \times \text{Mem} \times \mathbb{T} \\
\text{Result} & r & \in & \text{Result} \subseteq \text{ValCtx} \times \text{Mem} \times \mathbb{T} \\
\text{Tick} & \text{tick} & \in & \text{Tick} \subseteq \text{State} \times \text{Var} \times \text{Val} \to \mathbb{T} \\
\text{Context} & C & \to & [] \qquad \text{empty stack} \\
 & & | & (x, t) :: C \qquad \text{expression binding} \\
 & & | & (M, C) :: C \qquad \text{module binding} \\
\text{Value of expressions} & v & \to & \langle \lambda x.e, C \rangle \qquad \text{closure}
\end{array}
$$

Figure 3: Definition of the semantic domains in the concrete case.

To enforce that the tick function always produces fresh timestamps, the definition of the concrete State and Result ensure that the next timestamp that is recorded in memory is *larger* than any timestamp that is already used. Define:

$$
C < t \triangleq \begin{cases} \text{True} & C = [] \\ t' < t \wedge C' < t & C = (x, t') :: C' \\ C' < t \wedge C'' < t & C = (M, C') :: C'' \end{cases}
\qquad
V < t \triangleq \begin{cases} C < t & V = \langle \_, C \rangle \\ C < t & V = C \end{cases}
$$

$$
m < t \triangleq \forall t' \in \text{dom}(m) : t' < t \wedge m(t') < t
$$

and define:

$$
\text{State} \triangleq \{(C, m, t) | C < t \wedge m < t\} \qquad \text{Result} \triangleq \{(V, m, t) | V < t \wedge m < t\}
$$

Note that State $\subseteq$ Result, thus any statement about $r \in$ Result also holds for any $s \in$ State.

The fact that the t-component in any r bounds the C and m component means that the next location to be updated is fresh. After updating the memory, a new timestamp that is larger than the original t must be given by tick. This implies that: $\forall t : t < \text{tick}((\_, \_, t), \_, \_)$ should be required of tick. Moreover, for every concrete tick to have a corresponding abstract $\widehat{\text{tick}}$ connected through $\hat{\alpha}$, we want to require that: $\exists \widehat{\text{tick}} : \hat{\alpha} \circ \text{tick} = \widehat{\text{tick}} \circ \hat{\alpha}$, when:

$$
\hat{\alpha}(C) \triangleq \begin{cases} [] & C = [] \\ (x, \hat{\alpha}(t)) :: \hat{\alpha}(C') & C = (x, t) :: C' \\ (M, \hat{\alpha}(C')) :: \hat{\alpha}(C'') & C = (M, C') :: C'' \end{cases}
\qquad
\hat{\alpha}(V) \triangleq \begin{cases} \langle e, \hat{\alpha}(C) \rangle & V = \langle e, C \rangle \\ \hat{\alpha}(C) & V = C \end{cases}
$$

$$
\hat{\alpha}(m) \triangleq \bigcup_{t \in \text{dom}(m)} [\hat{\alpha}(t) \mapsto \{\hat{\alpha}(m(t))\}]
$$

Thus we can define:

$$\mathsf{Tick} \triangleq \{\mathsf{tick} | \forall t : t < \mathsf{tick}((\_,\_,t),\_,\_) \wedge \exists \widehat{\mathsf{tick}} : \hat{\alpha} \circ \mathsf{tick} = \widehat{\mathsf{tick}} \circ \hat{\alpha}\}$$

when the $\hat{\alpha}$ in $\widehat{\mathsf{tick}} \circ \hat{\alpha}$ maps over tuples and define:

$$\hat{\alpha}(\mathsf{tick}) \triangleq \widehat{\mathsf{tick}} \qquad \text{where } \hat{\alpha} \circ \mathsf{tick} = \widehat{\mathsf{tick}} \circ \hat{\alpha}$$

since if such a $\widehat{\mathsf{tick}}$ exists, it must be unique.

1.1.3 Operational Semantics

Now we give the operational semantics. The relation $\rightsquigarrow_{\mathsf{tick}}$ relates two machine states $\ell \in \mathsf{Left} \triangleq \mathsf{Expr} \times \mathsf{State}$ and $\rho \in \mathsf{Right} \triangleq \mathsf{Left} \uplus \mathsf{Result}$, using tick as the policy for memory allocation. Likewise, $\widehat{\rightsquigarrow}_{\widehat{\mathsf{tick}}}$ relates two machine states $\hat{\ell} \in \widehat{\mathsf{Left}} \triangleq \mathsf{Expr} \times \widehat{\mathsf{State}}$ and $\hat{\rho} \in \widehat{\mathsf{Right}} \triangleq \widehat{\mathsf{Left}} \uplus \widehat{\mathsf{Result}}$, using $\widehat{\mathsf{tick}}$.

There are auxiliary functions that are used to describe the relation. The addr and ctx operators each extract the address bound to a variable x and the context bound to a variable M from a context C.

$$\mathsf{addr}(C,x) \triangleq \begin{cases} \bot & C = [] \\ t & C = (x,t) :: C' \\ \mathsf{addr}(C',x) & C = (x',t) :: C' \wedge x' \neq x \\ \mathsf{addr}(C'',x) & C = (M,C') :: C'' \end{cases} \qquad \mathsf{ctx}(C,M) \triangleq \begin{cases} \bot & C = [] \\ C' & C = (M,C') :: C'' \\ \mathsf{ctx}(C'',M) & C = (M',C') :: C'' \wedge M' \neq M \\ \mathsf{ctx}(C',M) & C = (x,t) :: C' \end{cases}$$

Also, the update of the memory for the abstract operational semantics is not a strong update, which overwrites the previous value stored at the address that is written to.

$$\widehat{m}[\hat{t} \hat{\mapsto} \hat{v}](\widehat{t'}) \triangleq \begin{cases} \widehat{m}(\hat{t}) \cup \{\hat{v}\} & (\widehat{t'} = \hat{t}) \\ \widehat{m}(\widehat{t'}) & (\text{otherwise}) \end{cases}$$

We call this a "weak update" of the abstract memory.

The definitions for the operational semantics are given in Figure 4 and Figure 5.

1.1.4 Collecting Semantics

To define a semantics that is computable, we must formulate the collecting semantics as a least fixed point of a monotonic function that maps an element of some CPO D to D. In our case, $D \triangleq \wp((\mathsf{Left} \times \mathsf{Tick} \times \mathsf{Right}) \uplus (\mathsf{Right} \times \mathsf{Tick}))$. The semantics of an expression e starting from initial states in $S \subseteq \mathsf{State} \times \mathsf{Tick}$ is the collection of $\ell \rightsquigarrow_{\mathsf{tick}} \rho$ and (ρ, tick) derivable from initial configurations $((e,s),\mathsf{tick})$ with $(s,\mathsf{tick}) \in S$. Defining the transfer function is straightforward from the definition of the transition relation.

Definition 1.1 (Transfer function). Given $A \subseteq D$, define

$$\mathsf{Step}(A) \triangleq \left\{ \ell \rightsquigarrow_{\mathsf{tick}} \rho, (\rho,\mathsf{tick}) \middle| \frac{A'}{\ell \rightsquigarrow_{\mathsf{tick}} \rho} \wedge A' \subseteq A \wedge (\ell,\mathsf{tick}) \in A \right\}$$

The Step function is naturally monotonic, as a "cache" A that remembers more about the intermediate proof tree will derive more results than a cache that remembers less. Now, because of Tarski's fixpoint theorem, we can formulate the collecting semantics in fixpoint form.

Definition 1.2 (Collecting semantics). Given $e \in \mathsf{Expr}$ and $S \subseteq \mathsf{State} \times \mathsf{Tick}$, define:

$$[\![e]\!]S \triangleq \mathsf{lfp}(\lambda X.\mathsf{Step}(X) \cup \{((e,s),\mathsf{tick}) | (s,\mathsf{tick}) \in S\})$$

The abstract semantics is defined analogously. Let $\widehat{D} \triangleq \wp((\widehat{\mathsf{Left}} \times \widehat{\mathsf{Tick}} \times \widehat{\mathsf{Right}}) \uplus (\widehat{\mathsf{Right}} \times \widehat{\mathsf{Tick}}))$, then the transfer function and the semantics are defined so that they respect the Galois connection between $\wp(D)$ and $\wp(\widehat{D})$. How the CPOs are connected will be elaborated in the proof sketches.

Definition 1.3 (Abstract transfer function). Given $A^\# \subseteq \widehat{D}$, define

$$\mathsf{Step}^\#(A^\#) \triangleq \left\{ \hat{\ell} \widehat{\rightsquigarrow}_{\widehat{\mathsf{tick}}} \hat{\rho}, (\hat{\rho}, \widehat{\mathsf{tick}}) \middle| \frac{A'^\#}{\hat{\ell} \widehat{\rightsquigarrow}_{\widehat{\mathsf{tick}}} \hat{\rho}} \wedge A'^\# \subseteq A^\# \wedge (\hat{\ell}, \widehat{\mathsf{tick}}) \in A^\# \right\}$$

Definition 1.4 (Abstract semantics). Given $e \in \mathsf{Expr}$ and $S^\# \subseteq \widehat{\mathsf{State}} \times \widehat{\mathsf{Tick}}$, define:

$$[\![e]\!]^\# S^\# \triangleq \mathsf{lfp}(\lambda X^\#.\mathsf{Step}^\#(X^\#) \cup \{((e,\hat{s}),\widehat{\mathsf{tick}}) | (\hat{s},\widehat{\mathsf{tick}}) \in S^\#\})$$

$$\boxed{(e, C, m, t) \rightsquigarrow_{\mathsf{tick}} (V, m', t') \text{ or } (e', C', m', t')}$$

$$[\textsc{ExprID}]\ \frac{t_x = \mathsf{addr}(C, x) \qquad v = m(t_x)}{(x, C, m, t) \rightsquigarrow (v, m, t)} \qquad [\textsc{Fn}]\ \frac{}{(\lambda x.e, C, m, t) \rightsquigarrow (\langle \lambda x.e, C \rangle, m, t)}$$

$$[\textsc{AppL}]\ \frac{}{(e_1\ e_2, C, m, t) \rightsquigarrow (e_1, C, m, t)} \qquad [\textsc{AppR}]\ \frac{(e_1, C, m, t) \rightsquigarrow (\langle \lambda x.e_\lambda, C_\lambda \rangle, m_\lambda, t_\lambda)}{(e_1\ e_2, C, m, t) \rightsquigarrow (e_2, C, m_\lambda, t_\lambda)}$$

$$[\textsc{AppBody}]\ \frac{\begin{array}{c}(e_1, C, m, t) \rightsquigarrow (\langle \lambda x.e_\lambda, C_\lambda \rangle, m_\lambda, t_\lambda) \\ (e_2, C, m_\lambda, t_\lambda) \rightsquigarrow (v, m_a, t_a)\end{array}}{(e_1\ e_2, C, m, t) \rightsquigarrow (e_\lambda, (x, t_a) :: C_\lambda, m_a[t_a \mapsto v], \mathsf{tick}((C, m_a, t_a), x, v))}$$

$$[\textsc{App}]\ \frac{\begin{array}{c}(e_1, C, m, t) \rightsquigarrow (\langle \lambda x.e_\lambda, C_\lambda \rangle, m_\lambda, t_\lambda) \\ (e_2, C, m_\lambda, t_\lambda) \rightsquigarrow (v, m_a, t_a) \\ (e_\lambda, (x, t_a) :: C_\lambda, m_a[t_a \mapsto v], \mathsf{tick}((C, m_a, t_a), x, v)) \rightsquigarrow (v', m', t')\end{array}}{(e_1\ e_2, C, m, t) \rightsquigarrow (v', m', t')}$$

$$[\textsc{LinkL}]\ \frac{}{(e_1 \rtimes e_2, C, m, t) \rightsquigarrow (e_1, C, m, t)} \qquad [\textsc{LinkR}]\ \frac{(e_1, C, m, t) \rightsquigarrow (C', m', t')}{(e_1 \rtimes e_2, C, m, t) \rightsquigarrow (e_2, C', m', t')}$$

$$[\textsc{Link}]\ \frac{\begin{array}{c}(e_1, C, m, t) \rightsquigarrow (C', m', t') \\ (e_2, C', m', t') \rightsquigarrow (V, m'', t'')\end{array}}{(e_1 \rtimes e_2, C, m, t) \rightsquigarrow (V, m'', t'')} \qquad [\textsc{Empty}]\ \frac{}{(\varepsilon, C, m, t) \rightsquigarrow (C, m, t)} \qquad [\textsc{ModID}]\ \frac{C' = \mathsf{ctx}(C, M)}{(M, C, m, t) \rightsquigarrow (C', m, t)}$$

$$[\textsc{LetEL}]\ \frac{}{(\mathtt{let}\ x\ e_1\ e_2, C, m, t) \rightsquigarrow (e_1, C, m, t)}$$

$$[\textsc{LetER}]\ \frac{(e_1, C, m, t) \rightsquigarrow (v, m', t')}{(\mathtt{let}\ x\ e_1\ e_2, C, m, t) \rightsquigarrow (e_2, (x, t') :: C, m'[t' \mapsto v], \mathsf{tick}((C, m', t'), x, v))}$$

$$[\textsc{LetML}]\ \frac{}{(\mathtt{let}\ M\ e_1\ e_2, C, m, t) \rightsquigarrow (e_1, C, m, t)} \qquad [\textsc{LetMR}]\ \frac{(e_1, C, m, t) \rightsquigarrow (C', m', t')}{(\mathtt{let}\ M\ e_1\ e_2, C, m, t) \rightsquigarrow (e_2, (M, C') :: C, m', t')}$$

$$[\textsc{LetE}]\ \frac{\begin{array}{c}(e_1, C, m, t) \rightsquigarrow (v, m', t') \\ (e_2, (x, t') :: C, m'[t' \mapsto v], \mathsf{tick}((C, m', t'), x, v)) \rightsquigarrow (C', m'', t'')\end{array}}{(\mathtt{let}\ x\ e_1\ e_2, C, m, t) \rightsquigarrow (C', m'', t'')} \quad [\textsc{LetM}]\ \frac{\begin{array}{c}(e_1, C, m, t) \rightsquigarrow (C', m', t') \\ (e_2, (M, C') :: C, m', t') \rightsquigarrow (C'', m'', t'')\end{array}}{(\mathtt{let}\ M\ e_1\ e_2, C, m, t) \rightsquigarrow (C'', m'', t'')}$$

Figure 4: The concrete one-step transition relation. The subscript tick is omitted for brevity.

$\boxed{(e, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow}_{\widehat{\mathsf{tick}}} (\widehat{V}, \widehat{m'}, \hat{t'}) \text{ or } (e', \widehat{C'}, \widehat{m'}, \hat{t'})}$

$$[\textsc{ExprID}]\ \frac{\hat{t}_x = \mathsf{addr}(\widehat{C}, x) \qquad \hat{v} \in \widehat{m}(\hat{t}_x)}{(x, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\hat{v}, \widehat{m}, \hat{t})} \qquad [\textsc{Fn}]\ \frac{}{(\lambda x.e, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\langle \lambda x.e, \widehat{C} \rangle, \widehat{m}, \hat{t})}$$

$$[\textsc{AppL}]\ \frac{}{(e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (e_1, \widehat{C}, \widehat{m}, \hat{t})} \qquad [\textsc{AppR}]\ \frac{(e_1, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\langle \lambda x.e_\lambda, \widehat{C}_\lambda \rangle, \widehat{m}_\lambda, \hat{t}_\lambda)}{(e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (e_2, \widehat{C}, \widehat{m}_\lambda, \hat{t}_\lambda)}$$

$$[\textsc{AppBody}]\ \frac{\begin{array}{c}(e_1, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\langle \lambda x.e_\lambda, \widehat{C}_\lambda \rangle, \widehat{m}_\lambda, \hat{t}_\lambda) \\ (e_2, \widehat{C}, \widehat{m}_\lambda, \hat{t}_\lambda) \widehat{\rightsquigarrow} (v, \widehat{m}_a, \hat{t}_a)\end{array}}{(e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (e_\lambda, (x, \hat{t}_a) :: \widehat{C}_\lambda, \widehat{m}_a[\hat{t}_a \hat{\mapsto} \hat{v}], \widehat{\mathsf{tick}}((\widehat{C}, \widehat{m}_a, \hat{t}_a), x, \hat{v}))}$$

$$[\textsc{App}]\ \frac{\begin{array}{c}(e_1, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\langle \lambda x.e_\lambda, \widehat{C}_\lambda \rangle, \widehat{m}_\lambda, \hat{t}_\lambda) \\ (e_2, \widehat{C}, \widehat{m}_\lambda, \hat{t}_\lambda) \widehat{\rightsquigarrow} (\hat{v}, \widehat{m}_a, \hat{t}_a) \\ (e_\lambda, (x, \hat{t}_a) :: \widehat{C}_\lambda, \widehat{m}_a[\hat{t}_a \hat{\mapsto} \hat{v}], \widehat{\mathsf{tick}}((\widehat{C}, \widehat{m}_a, \hat{t}_a), x, \hat{v})) \widehat{\rightsquigarrow} (\hat{v'}, \widehat{m'}, \hat{t'})\end{array}}{(e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\hat{v'}, \widehat{m'}, \hat{t'})}$$

$$[\textsc{LinkL}]\ \frac{}{(e_1 \rtimes e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (e_1, \widehat{C}, \widehat{m}, \hat{t})} \qquad [\textsc{LinkR}]\ \frac{(e_1, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\widehat{C'}, \widehat{m'}, \hat{t'})}{(e_1 \rtimes e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (e_2, \widehat{C'}, \widehat{m'}, \hat{t'})}$$

$$[\textsc{Link}]\ \frac{\begin{array}{c}(e_1, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\widehat{C'}, \widehat{m'}, \hat{t'}) \\ (e_2, \widehat{C'}, \widehat{m'}, \hat{t'}) \widehat{\rightsquigarrow} (\widehat{V}, \widehat{m''}, \hat{t''})\end{array}}{(e_1 \rtimes e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\widehat{V}, \widehat{m''}, \hat{t''})} \qquad [\textsc{Empty}]\ \frac{}{(\varepsilon, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\widehat{C}, \widehat{m}, \hat{t})}$$

$$[\textsc{ModID}]\ \frac{\widehat{C'} = \mathsf{ctx}(\widehat{C}, M)}{(M, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\widehat{C'}, \widehat{m}, \hat{t})} \qquad [\textsc{LetEL}]\ \frac{}{(\mathtt{let}\ x\ e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (e_1, \widehat{C}, \widehat{m}, \hat{t})}$$

$$[\textsc{LetER}]\ \frac{(e_1, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\hat{v}, \widehat{m'}, \hat{t'})}{(\mathtt{let}\ x\ e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (e_2, (x, \hat{t'}) :: \widehat{C}, \widehat{m'}[\hat{t'} \hat{\mapsto} \hat{v}], \widehat{\mathsf{tick}}((\widehat{C}, \widehat{m'}, \hat{t'}), x, \hat{v}))}$$

$$[\textsc{LetML}]\ \frac{}{(\mathtt{let}\ M\ e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (e_1, \widehat{C}, \widehat{m}, \hat{t})} \qquad [\textsc{LetMR}]\ \frac{(e_1, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\widehat{C'}, \widehat{m'}, \hat{t'})}{(\mathtt{let}\ M\ e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (e_2, (M, \widehat{C'}) :: \widehat{C}, \widehat{m'}, \hat{t'})}$$

$$[\textsc{LetE}]\ \frac{\begin{array}{c}(e_1, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\hat{v}, \widehat{m'}, \hat{t'}) \\ (e_2, (x, \hat{t'}) :: \widehat{C}, \widehat{m'}[\hat{t'} \hat{\mapsto} \hat{v}], \widehat{\mathsf{tick}}((\widehat{C}, \widehat{m'}, \hat{t'}), x, \hat{v})) \widehat{\rightsquigarrow} (\widehat{C'}, \widehat{m''}, \hat{t''})\end{array}}{(\mathtt{let}\ x\ e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\widehat{C'}, \widehat{m''}, \hat{t''})}$$

$$[\textsc{LetM}]\ \frac{\begin{array}{c}(e_1, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\widehat{C'}, \widehat{m'}, \hat{t'}) \\ (e_2, (M, \widehat{C'}) :: \widehat{C}, \widehat{m'}, \hat{t'}) \widehat{\rightsquigarrow} (\widehat{C''}, \widehat{m''}, \hat{t''})\end{array}}{(\mathtt{let}\ M\ e_1\ e_2, \widehat{C}, \widehat{m}, \hat{t}) \widehat{\rightsquigarrow} (\widehat{C''}, \widehat{m''}, \hat{t''})}$$

Figure 5: The abstract one-step transition relation. The subscript $\widehat{\mathsf{tick}}$ is omitted for brevity.

1.2 Proof Sketches

1.2.1 Well-Definedness of the Operational Semantics

The Left and Right sets that are connected by the concrete transition relation $\rightsquigarrow_{\mathsf{tick}}$ are restricted by the property that the timestamps of each element are bound by its time component. Therefore, for the concrete operational semantics to be truly well-defined, we have to check whether our inductive definition preserves the property of time-boundedness. This is a simple proof by induction on the transition relation, since tick always produces increasing timestamps.

Claim 1.1 (Well-definedness of $\rightsquigarrow_{\mathsf{tick}}$). For all e, C, m, t and $\rho \in \mathrm{Expr} \times \mathrm{Ctx} \times \mathrm{Mem} \times \mathbb{T} \uplus \mathrm{ValCtx} \times \mathrm{Mem} \times \mathbb{T}$, if $s \triangleq (C, m, t) \in \mathrm{State}$ and $(e, s) \rightsquigarrow_{\mathsf{tick}} \rho$ according to the inference rules in Figure 4, we have that $\rho \in \mathrm{Right}$.

Sketch. Induction on the definition of $\rightsquigarrow_{\mathsf{tick}}$. □

We also have to check that $\hat{\alpha}(\mathsf{tick})$ is well-defined.

Claim 1.2 (Well-definedness of $\hat{\alpha}(\mathsf{tick})$). For all $\mathsf{tick} \in \mathrm{State} \times \mathrm{Var} \times \mathrm{Val} \to \mathbb{T}$, if $\widehat{\mathsf{tick}_1} \circ \hat{\alpha} = \hat{\alpha} \circ \mathsf{tick}$ and $\widehat{\mathsf{tick}_2} \circ \hat{\alpha} = \hat{\alpha} \circ \mathsf{tick}$, we have that $\widehat{\mathsf{tick}_1} = \widehat{\mathsf{tick}_2}$.

Sketch. We have to prove that for all $\hat{s} \in \widehat{\mathrm{State}}$, $x \in \mathrm{Var}$, $\hat{v} \in \widehat{\mathrm{Val}}$, $\widehat{\mathsf{tick}_1}(\hat{s}, x, \hat{v}) = \widehat{\mathsf{tick}_2}(\hat{s}, x, \hat{v})$. This is true if for all $\hat{s} \in \widehat{\mathrm{State}}$, there exists a $s \in \mathrm{State}$ such that $\hat{\alpha}(s) = \hat{s}$. Then $\widehat{\mathsf{tick}_1}(\hat{s}, x, \hat{v}) = \mathsf{tick}(\hat{\alpha}(s), x, \hat{\alpha}(v)) = \widehat{\mathsf{tick}_2}(\hat{s}, x, \hat{v})$ by assumption, thus the desired conclusion is proved. The statement $\forall \hat{s} \in \widehat{\mathrm{State}} : \exists s \in \mathrm{State} : \hat{\alpha}(s) = \hat{s}$ can be proved by using $\hat{\alpha}^{-1}$ to generate fresh timestamps for all abstract times in $\hat{s}$, and induction on the number of timestamps in $\hat{s}$. Note that the time-boundedness of $s \in \mathrm{State}$ is ensured if we use the maximum of all generated timestamps as the first argument to $\hat{\alpha}^{-1}$ and the time component of $\hat{s}$ as the second argument. □

We finally check that for all $\widehat{\mathsf{tick}}$, there exists a tick such that $\hat{\alpha}(\mathsf{tick}) = \widehat{\mathsf{tick}}$.

Claim 1.3 (Well-definedness of Tick). For all $\widehat{\mathsf{tick}} \in \widehat{\mathrm{Tick}}$, there exists a $\mathsf{tick} \in \mathrm{Tick}$ such that $\hat{\alpha}(\mathsf{tick}) = \widehat{\mathsf{tick}}$.

Sketch. Let $\mathsf{tick}((C, m, t), x, v) \triangleq \hat{\alpha}^{-1}(t, \widehat{\mathsf{tick}}((\hat{\alpha}(C), \hat{\alpha}(m), \hat{\alpha}(t)), x, \hat{\alpha}(v)))$. Then $\mathsf{tick} \in \mathrm{Tick}$ by the properties of $\hat{\alpha}^{-1}$. □

1.2.2 Galois Connection between $\wp(D)$ and $\wp(\widehat{D})$

We understand by $\hat{\alpha}(\ell) = (e, \hat{\alpha}(s))$, when $\ell = (e, s) \in \mathrm{Left}$ and $\hat{\alpha}(s)$ maps over all coordinates. Likewise, we understand $\hat{\alpha}(\rho)$ to be $\hat{\alpha}(\ell)$ when $\rho = \ell$ and $\hat{\alpha}(r)$ when $\rho = r$ and $\hat{\alpha}(r)$ maps over all coordinates.

Definition 1.5 (Abstraction and Concretization). Define $\alpha : \wp(D) \to \wp(\widehat{D})$ and $\gamma : \wp(\widehat{D}) \to \wp(D)$ by:

$$\alpha(A) \triangleq \{\hat{\alpha}(\ell) \mathrel{\widehat{\rightsquigarrow}}_{\hat{\alpha}(\mathsf{tick})} \hat{\alpha}(\rho) | \ell \rightsquigarrow_{\mathsf{tick}} \rho \in A\} \cup \{(\hat{\alpha}(\rho), \hat{\alpha}(\mathsf{tick})) | (\rho, \mathsf{tick}) \in A\}$$

$$\gamma(A^\#) \triangleq \{\ell \rightsquigarrow_{\mathsf{tick}} \rho | \hat{\alpha}(\ell) \mathrel{\widehat{\rightsquigarrow}}_{\hat{\alpha}(\mathsf{tick})} \hat{\alpha}(\rho) \in A^\#\} \cup \{(\rho, \mathsf{tick}) | (\hat{\alpha}(\rho), \hat{\alpha}(\mathsf{tick})) \in A^\#\}$$

Then it is straightforward to see that:

Claim 1.4 (Galois Connection). $\wp(D) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \wp(\widehat{D})$. That is, $\forall A \subseteq D, A^\# \subseteq \widehat{D} : \alpha(A) \subseteq A^\# \Leftrightarrow A \subseteq \gamma(A^\#)$.

Sketch. Straightforward from the definitions of α and γ. □

Also, we can show that the concrete and abstract semantic functions are soundly connected:

Claim 1.5 (Sound Abstraction). For all ℓ, tick, and ρ, $\ell \rightsquigarrow_{\mathsf{tick}} \rho$ implies $\hat{\alpha}(\ell) \mathrel{\widehat{\rightsquigarrow}}_{\hat{\alpha}(\mathsf{tick})} \hat{\alpha}(\rho)$. That is, $\alpha \circ \mathsf{Step} \subseteq \mathsf{Step}^\# \circ \alpha$ and thus $\alpha([\![e]\!]S) \subseteq [\![e]\!]^\# \alpha(S)$ for all e and $S \subseteq \mathrm{State} \times \mathrm{Tick}$.

Sketch. Induction on the definition of $\rightsquigarrow_{\mathsf{tick}}$. □

1.2.3 Computability of the Abstract Semantics

Now we can say that $[\![e]\!]^\# \alpha(S)$ is a sound abstraction of $[\![e]\!]S$. However, is it true that $[\![e]\!]^\# \alpha(S)$ is finitely computable? Note that in practice, all reachable configurations are derived from some expression e evaluated from the empty context $[]$ and empty memory $\emptyset$. We claim that in such situations, when the abstract semantics is computed from a finite set $S^\#$ of initial states, the resulting computation $[\![e]\!]^\# S^\#$ has finite cardinality.

Since $\widehat{\mathbb{T}}$ is finite, all we have to prove is that all reachable *signatures* are finite. What we mean by a *signature* is a context that is stripped of all timestamps. Explicitly, we mean an element of an inductively defined set Sig given by $X \to [] \mid x :: X \mid (M, X) :: X$. Then we may inductively define $\lfloor C \rfloor$ and $\lfloor \widehat{C} \rfloor$ to be the signatures that are obtained by stripping all timestamps from C and $\widehat{C}$. Moreover, we may define $\lfloor m \rfloor \triangleq \{\lfloor C \rfloor | \exists t : \langle \_, C \rangle = m(t)\}$ and $\lfloor \widehat{m} \rfloor \triangleq \{\lfloor \widehat{C} \rfloor | \exists \hat{t} :$

$\langle \_, \widehat{C} \rangle \in \widehat{m}(\hat{t})\}$ to be all signatures in a memory. Finally, we may define $\lfloor \rho \rfloor$ as the union of $\lfloor C \rfloor$ and $\lfloor m \rfloor$, when C is the context component of ρ and m is the memory component of ρ. $\lfloor \hat{\rho} \rfloor$ can be analogously defined.

If we can prove that for all e and $\hat{s}$, $\bigcup_{(e,\hat{s})\widehat{\rightsquigarrow}^* \hat{\rho}} \lfloor \hat{\rho} \rfloor \subseteq X$, when X is a finite set containing all reachable signatures from $(e, \hat{s})$, we can show that all reachable $\hat{\rho}$s are finite, since $\widehat{\mathbb{T}}$ is finite. It turns out, since the signature of the modules that are pushed into the stack C can be accurately inferred from the definition of the operational semantics, we can *compute* such an X. Thus we have:

Claim 1.6 (Computability of the Abstract Semantics). If $S^\#$ is finite, $[\![e]\!]^\# S^\#$ is finite.

Sketch. By existence of a function that computes all reachable signatures and by induction on $\widehat{\rightsquigarrow}$.

To elaborate, let this function be called f. We prove two statements, the first being $\forall \hat{\rho} : \lfloor \hat{\rho} \rfloor \subseteq f(\hat{\rho})$ and the second being $\forall \hat{\ell}, \hat{\rho} : \hat{\ell} \widehat{\rightsquigarrow} \hat{\rho} \Rightarrow f(\hat{\rho}) \subseteq f(\hat{\ell})$. Then for all $\hat{\rho}$ such that $(e, \hat{s})\widehat{\rightsquigarrow}^* \hat{\rho}$, $\lfloor \hat{\rho} \rfloor \subseteq f(\rho) \subseteq f(\ell)$, thus all signatures in $[\![e]\!]^\# S^\#$ can be bound by $\{f(e, \hat{s}) | (\hat{s}, \_) \in S\}$. □

We stress that this is a nontrivial result, since our definition of C allows contexts to be pushed into the stack. In a language where functors that take modules that are not annotated by signatures as arguments, this claim does not hold.

2 Results on Equivalence

In this section, we define what it means for semantics that use different timestamps to be *equivalent*. Our framework hinges heavily on the definition of equivalence, since when we link semantics that use two different timestamps, we end up with a semantics that uses a totally different $\mathbb{T}$ and tick. Even in the case without linking, we need to justify why no matter our choice of $(\mathbb{T}, \leq, \widehat{\mathbb{T}}, \hat{\alpha})$, the analysis overapproximates a *compatible* notion of execution.

In this section, we assume a pair of semantics, each parametrized with $(\mathbb{T}, \leq, \widehat{\mathbb{T}}, \hat{\alpha})$ and $(\mathbb{T}', \leq', \widehat{\mathbb{T}}', \hat{\alpha}')$.

2.1 Definitions

We first define what it means for two states $s \in \text{State}$ and $s' \in \text{State}'$ to be equivalent. Note that $s = (C, m, t)$ and $s' = (C', m', t')$ for some contexts C, C', some memories m, m', and some times t, t'. The choice of t and t' is "not special" in the sense that as long as they bound the context and memory, tick will continue producing fresh addresses. Thus, the notion of equivalence is defined by how the "information extractable" from the C and m components are "same".

Note that the information that is extractable are only accessed through a sequence of names x and M. Thus, one may imagine access "paths" with names on the edges and information sources(C and t) on the nodes. Then the definition of equivalence may be given as equivalence on all access paths with same labels on edges. Taking this a step further, one may imagine an "access graph" that collects all reachable C and t as the nodes of the graph with directed edges connecting the nodes corresponding to accesses by names or the expression part of closures in memory.

$$
\begin{array}{rcl}
\text{Step}_m(G) & \triangleq & \{C \xrightarrow{x} t, t | C \in G \wedge t = \text{addr}(C, x)\} \\
& \cup & \{C \xrightarrow{M} C', C' | C \in G \wedge C' = \text{ctx}(C, M)\} \\
& \cup & \{t \xrightarrow{e} C, C | t \in G \wedge \langle e, C \rangle = m(t)\} \\
\underline{C, m} & \triangleq & \text{lfp}(\lambda X.\text{Step}_m(X) \cup \{C\})
\end{array}
\qquad
\begin{array}{rcl}
\widehat{\text{Step}}_{\widehat{m}}(\widehat{G}) & \triangleq & \{\widehat{C} \xrightarrow{x} \hat{t}, \hat{t} | \widehat{C} \in \widehat{G} \wedge \hat{t} = \text{addr}(\widehat{C}, x)\} \\
& \cup & \{\widehat{C} \xrightarrow{M} \widehat{C}', \widehat{C}' | \widehat{C} \in \widehat{G} \wedge \widehat{C}' = \text{ctx}(\widehat{C}, M)\} \\
& \cup & \{\hat{t} \xrightarrow{e} \widehat{C}, \widehat{C} | \hat{t} \in \widehat{G} \wedge \langle e, \widehat{C} \rangle \in \widehat{m}(\hat{t})\} \\
\underline{\widehat{C}, \widehat{m}} & \triangleq & \text{lfp}(\lambda \widehat{X}.\widehat{\text{Step}}_{\widehat{m}}(\widehat{X}) \cup \{\widehat{C}\})
\end{array}
$$

The graphs $\underline{C, m}$ and $\underline{\widehat{C}, \widehat{m}}$ are *rooted labelled directed graph*s, with the initial context as the root. Then the definition of equivalence between states simply means that the access graphs are isomorphic.

Definition 2.1 (Equivalent Concrete States). Let $s = (C, m, t) \in \text{State}$ and $s' = (C', m', t') \in \text{State}'$. We say s is *equivalent* to s' and write $s \cong s'$ when there exists a $\varphi : \text{Ctx} \uplus \mathbb{T} \to \text{Ctx}' \uplus \mathbb{T}'$ and $\varphi^{-1} : \text{Ctx}' \uplus \mathbb{T}' \to \text{Ctx} \uplus \mathbb{T}$ such that:

1. $\varphi(C) = C'$ and $\varphi^{-1}(C') = C$
2. $\forall \text{node}_1, \text{node}_2 : \text{node}_1 \xrightarrow{\text{lbl}} \text{node}_2 \in \underline{C, m} \Rightarrow \varphi(\text{node}_1) \xrightarrow{\text{lbl}} \varphi(\text{node}_2) \in \underline{C', m'}$
3. $\forall \text{node}_1', \text{node}_2' : \text{node}_1' \xrightarrow{\text{lbl}} \text{node}_2' \in \underline{C', m'} \Rightarrow \varphi^{-1}(\text{node}_1') \xrightarrow{\text{lbl}} \varphi^{-1}(\text{node}_2') \in \underline{C, m}$
4. $\forall \text{node}, \text{node}' : \text{node} \in \underline{C, m} \Rightarrow \varphi^{-1}(\varphi(\text{node})) = \text{node}$ and $\text{node}' \in \underline{C', m'} \Rightarrow \varphi(\varphi^{-1}(\text{node}')) = \text{node}'$

Definition 2.2 (Equivalent Abstract States). Let $\hat{s} = (\widehat{C}, \widehat{m}, \hat{t}) \in \widehat{\text{State}}$ and $\hat{s}' = (\widehat{C}', \widehat{m}', \hat{t}') \in \widehat{\text{State}}'$. We say $\hat{s}$ is *equivalent* to $\hat{s}'$ and write $\hat{s} \widehat{\cong} \hat{s}'$ when there exists a $\hat{\varphi} : \widehat{\text{Ctx}} \uplus \widehat{\mathbb{T}} \to \widehat{\text{Ctx}}' \uplus \widehat{\mathbb{T}}'$ and $\hat{\varphi}^{-1} : \widehat{\text{Ctx}}' \uplus \widehat{\mathbb{T}}' \to \widehat{\text{Ctx}} \uplus \widehat{\mathbb{T}}$ such that:

1. $\hat{\varphi}(\widehat{C}) = \widehat{C'}$ and $\hat{\varphi}^{-1}(\widehat{C'}) = \widehat{C}$
2. $\forall \text{node}_1, \text{node}_2 : \text{node}_1 \xrightarrow{\text{lbl}} \text{node}_2 \in \underline{\widehat{C}, \widehat{m}} \Rightarrow \hat{\varphi}(\text{node}_1) \xrightarrow{\text{lbl}} \hat{\varphi}(\text{node}_2) \in \underline{\widehat{C'}, \widehat{m'}}$
3. $\forall \text{node}_1'.\text{node}_2' : \text{node}_1' \xrightarrow{\text{lbl}} \text{node}_2' \in \underline{\widehat{C'}, \widehat{m'}} \Rightarrow \hat{\varphi}^{-1}(\text{node}_1') \xrightarrow{\text{lbl}} \hat{\varphi}^{-1}(\text{node}_2') \in \underline{\widehat{C}, \widehat{m}}$
4. $\forall \text{node}, \text{node}' : \text{node} \in \underline{\widehat{C}, \widehat{m}} \Rightarrow \hat{\varphi}^{-1}(\hat{\varphi}(\text{node})) = \text{node}$ and $\text{node}' \in \underline{\widehat{C'}, \widehat{m'}} \Rightarrow \hat{\varphi}(\hat{\varphi}^{-1}(\text{node}')) = \text{node}'$

We also define equivalence between results $r = (V, m.t) \in \text{Result}$ and $r' = (V', m', t') \in \text{Result}'$ by the conjunction of the equality between the expression part of V and the isomorphism between access graphs. Also, equivalence between $\ell = (e, s) \in \text{Left}$ and $\ell' = (e', s') \in \text{Left}'$ can be similarly defined as $e = e' \wedge s \cong s'$. Thus equivalence between $\rho \in \text{Right}$ and $\rho' \in \text{Right}'$ is defined, as either $\rho \in \text{Left}$ or $\rho \in \text{Result}$.

Then the equivalence between elements of $\wp(D)$ and $\wp(D')$, the CPOs, can be defined as:

Definition 2.3 (Equivalence between Elements of $\wp(D)$ and $\wp(D')$)**.** Let $A \subseteq D$ and $A' \subseteq D'$. We say that A and A' are equivalent and write $A \cong A'$ iff:

1. $\forall \ell, \text{tick}, \rho : \ell \rightsquigarrow_{\text{tick}} \rho \in A \Rightarrow \exists \ell', \text{tick}', \rho' : \ell' \rightsquigarrow_{\text{tick}'} \rho' \in A' \wedge \ell \cong \ell' \wedge \rho \cong \rho'$
2. $\forall \ell', \text{tick}', \rho' : \ell' \rightsquigarrow_{\text{tick}'} \rho' \in A' \Rightarrow \exists \ell, \text{tick}, \rho : \ell \rightsquigarrow_{\text{tick}} \rho \in A \wedge \ell \cong \ell' \wedge \rho \cong \rho'$
3. $\forall \rho, \text{tick} : (\rho, \text{tick}) \in A \Rightarrow \exists \rho', \text{tick}' : (\rho', \text{tick}') \in A' \wedge \rho \cong \rho'$
4. $\forall \rho', \text{tick}' : (\rho', \text{tick}') \in A' \Rightarrow \exists \rho, \text{tick} : (\rho, \text{tick}) \in A \wedge \rho \cong \rho'$

Likewise, we can define equivalence between elements of $\wp(\widehat{D})$ and $\wp(\widehat{D'})$.

Definition 2.4 (Equivalence between Elements of $\wp(\widehat{D})$ and $\wp(\widehat{D'})$)**.** Let $A^\# \subseteq \widehat{D}$ and $A'^\# \subseteq \widehat{D'}$. We say that $A^\#$ and $A'^\#$ are equivalent and write $A^\# \widehat{\cong}^\# A'^\#$ iff:

1. $\forall \hat{\ell}, \widehat{\text{tick}}, \hat{\rho} : \hat{\ell} \,\widehat{\rightsquigarrow}_{\widehat{\text{tick}}}\, \hat{\rho} \in A^\# \Rightarrow \exists \hat{\ell}', \widehat{\text{tick}}', \hat{\rho}' : \hat{\ell}' \,\widehat{\rightsquigarrow}_{\widehat{\text{tick}}'}\, \hat{\rho}' \in A'^\# \wedge \hat{\ell} \widehat{\cong} \hat{\ell}' \wedge \hat{\rho} \widehat{\cong} \hat{\rho}'$
2. $\forall \hat{\ell}', \widehat{\text{tick}}', \hat{\rho}' : \hat{\ell}' \,\widehat{\rightsquigarrow}_{\widehat{\text{tick}}'}\, \hat{\rho}' \in A'^\# \Rightarrow \exists \hat{\ell}, \widehat{\text{tick}}, \hat{\rho} : \hat{\ell} \,\widehat{\rightsquigarrow}_{\widehat{\text{tick}}}\, \hat{\rho} \in A^\# \wedge \hat{\ell} \widehat{\cong} \hat{\ell}' \wedge \hat{\rho} \widehat{\cong} \hat{\rho}'$
3. $\forall \hat{\rho}, \widehat{\text{tick}} : (\hat{\rho}, \widehat{\text{tick}}) \in A^\# \Rightarrow \exists \hat{\rho}', \widehat{\text{tick}}' : (\hat{\rho}', \widehat{\text{tick}}') \in A' \wedge \hat{\rho} \widehat{\cong} \hat{\rho}'$
4. $\forall \hat{\rho}', \widehat{\text{tick}}' : (\hat{\rho}', \widehat{\text{tick}}') \in A'^\# \Rightarrow \exists \hat{\rho}, \widehat{\text{tick}} : (\hat{\rho}, \widehat{\text{tick}}) \in A \wedge \hat{\rho} \widehat{\cong} \hat{\rho}'$

2.2 Proof Sketches

2.2.1 Evaluation Preserves Equivalence

To prove if we actually did define equivalence sensibly, we must show that the operational semantics preserves equivalence. That is, we need to prove that starting from equivalent configurations, we end up in equivalent configurations.

Claim 2.1 (Evaluation Preserves Equivalence)**.** For all $\ell \in \text{Left}$, $\text{tick} \in \text{Tick}$, $\rho \in \text{Right}$, $\ell' \in \text{Left}'$, $\text{tick}' \in \text{Tick}'$,

$$\ell \rightsquigarrow_{\text{tick}} \rho \wedge \ell \cong \ell' \Rightarrow \exists \rho' : \ell' \rightsquigarrow_{\text{tick}'} \rho' \wedge \rho \cong \rho'$$

Thus, if $S \subseteq \text{State} \times \text{Tick}$ and $S' \subseteq \text{State}' \times \text{Tick}'$ are equivalent, $[\![e]\!]S \cong [\![e]\!]S'$.

Sketch. To perform induction on $\rightsquigarrow_{\text{tick}}$, we need to strengthen the claim. For convenience, we write $(\rho, \varphi) \cong (\rho', \varphi^{-1})$ to emphasize that the graph isomorphism is given by φ.

Then we can strengthen the claim to a claim about graph isomorphisms.

$$\begin{aligned}&\forall \ell, \text{tick}, \rho, \ell', \text{tick}', \varphi, \varphi^{-1} : \ell \rightsquigarrow_{\text{tick}} \rho \wedge (\ell, \varphi) \cong (\ell', \varphi^{-1}) \Rightarrow \\ &\exists \rho', \phi, \phi^{-1} : \ell' \rightsquigarrow_{\text{tick}'} \rho' \wedge (\rho, \phi) \cong (\rho', \phi^{-1}) \wedge \varphi(n)|_{n<t} = \phi(n)|_{n<t} \\ &\text{when } t \text{ is the time component of } \ell\end{aligned}$$

The important part here is that ϕ is an *extension* of φ in the sense that it agrees with φ with nodes in $\underline{\ell}$. Thus the induction hypothesis will push through. □

2.2.2 Concretization Preserves Equivalence

For the definition of equivalence to be compatible with analysis, we want to show that if the abstract initial states are equivalent, so are the concretization of those states. If this is true, we can obtain an overapproximation of an equivalent semantics by $[\![e]\!]\gamma(S^\#) \cong [\![e]\!]\gamma'(S'^\#) \subseteq \gamma'([\![e]\!]^\# S'^\#)$ from $S^\# \cong^\# S'^\#$. The first $\cong$ is from the fact that evaluation preserves equivalence and concretization preserves equivalence. The second $\subseteq$ is from the fact that $\mathsf{Step}^\#$ is a sound approximation of Step. The claim we want to prove is: If $S^\# \cong^\# S'^\#$, then $\gamma(S^\#) \cong \gamma'(S'^\#)$.

Claim 2.2 (Concretization Preserves Equivalence). For all $S^\# \subseteq \widehat{\mathrm{State}} \times \widehat{\mathrm{Tick}}$ and $S'^\# \subseteq \widehat{\mathrm{State}}' \times \widehat{\mathrm{Tick}}'$, $S^\# \cong^\# S'^\#$ implies $\gamma(S^\#) \cong \gamma'(S'^\#)$.

Sketch. We want to prove:

$$\forall s \in \mathrm{State}, \hat{s}' \in \widehat{\mathrm{State}}' \ : \ \hat{\alpha}(s) \mathrel{\widehat{\cong}} \hat{s}' \Rightarrow \exists s' \in \widehat{\mathrm{State}}' \ : \ s \cong s' \wedge \hat{\alpha}'(s') = \hat{s}'$$

If this is true, $\forall s \in \gamma(s^\#) \ : \ \exists s' \in \gamma(s'^\#) \ : \ s \cong s'$.

Proving the same statement in the opposite side leads to $\forall s' \in \gamma(s'^\#) \ : \ \exists s \in \gamma(s^\#) \ : \ s \cong s'$, so that $\gamma(S^\#) \cong \gamma'(S'^\#)$. The proof of the above statement involves constructing such a s' by traversal over reachable subparts n of s, (1) translating n to a reachable part $\hat{n}'$ of $\hat{s}'$ by $\hat{\varphi} \circ \hat{\alpha}$(when $\hat{\varphi}$ is the isomorphism between $\hat{\alpha}(s)$ and $\hat{s}'$) and (2) lifting $\hat{n}'$ to a reachable part n' of s' while tabulating the graph isomorphism φ between s and s'. Finally, (3) the unreachable parts of s' are filled in with dummy values that translate to the unfilled parts of $\hat{s}'$. □

3 Results on Linking

In this section, we define how linking between different time domains is performed. First, we assume two time domains $(\mathbb{T}_1, \leq_1, \widehat{\mathbb{T}}_1, \hat{\alpha}_1)$ and $(\mathbb{T}_2, \leq_2, \widehat{\mathbb{T}}_2, \hat{\alpha}_2)$. Then we can define $(\mathbb{T}_+, \leq_+, \widehat{\mathbb{T}}_+, \hat{\alpha}_+)$ by:

$$\mathbb{T}_+ \triangleq 2\mathbb{Z} \times \mathbb{T}_1 \uplus (2\mathbb{Z}+1) \times \mathbb{T}_2 \quad t_+ \leq_+ t'_+ \triangleq \text{lexicographic} \quad \widehat{\mathbb{T}}_+ \triangleq \widehat{\mathbb{T}}_1 + \widehat{\mathbb{T}}_2 \quad \hat{\alpha}_+(t_+) \triangleq \begin{cases} (0, \hat{\alpha}_1(t_1)) & t_+ = (2n, t_1) \\ (1, \hat{\alpha}_2(t_2)) & t_+ = (2n+1, t_2) \end{cases}$$

when $X + Y$ is the separated sum of X and Y. A $\hat{\alpha}_+^{-1} \in \mathbb{T}_+ \times \widehat{\mathbb{T}}_+ \to \mathbb{T}_+$ that satisfies the two conditions can be constructed by using some $\hat{\alpha}_1^{-1}$, $\hat{\alpha}_2^{-1}$, $0_1 \in \mathbb{T}_1$ and $0_2 \in \mathbb{T}_2$:

$$\hat{\alpha}_+^{-1}(t_+, \hat{t}_+) \triangleq \begin{cases} (2n, \hat{\alpha}_1^{-1}(t_1, \hat{t}_1)) & t_+ = (2n, t_1) \wedge \hat{t}_+ = \hat{t}_1 \\ (2n+1, \hat{\alpha}_2^{-1}(t_2, \hat{t}_2)) & t_+ = (2n+1, t_2) \wedge \hat{t}_+ = \hat{t}_2 \\ (2n+1, \hat{\alpha}_2^{-1}(0_2, \hat{t}_2)) & t_+ = (2n, t_1) \wedge \hat{t}_+ = \hat{t}_2 \\ (2n+2, \hat{\alpha}_1^{-1}(0_1, \hat{t}_1)) & t_+ = (2n+1, t_2) \wedge \hat{t}_+ = \hat{t}_1 \end{cases}$$

Thus $(\mathbb{T}_+, \leq_+, \widehat{\mathbb{T}}_+, \hat{\alpha}_+)$ satisfies our requirements.

The rest of this section explains how, given two initial conditions $S_1 \subseteq \mathrm{State}_1 \times \mathrm{Tick}_1$ and $S_2 \subseteq \mathrm{State}_2 \times \mathrm{Tick}_2$, the semantics starting from the *injected* initial state $S_1 \rhd S_2$ is equal to S_1 *linked* with the semantics computed in advance from S_2. That is, we want to prove:

$$[\![e]\!](S_1 \rhd S_2) = S_1 \bowtie [\![e]\!] S_2$$

and

$$[\![e]\!]^\#(S_1^\# \rhd^\# S_2^\#) = S_2^\# \bowtie^\# [\![e]\!]^\# S_2^\#$$

Before descending into details, we want to clarify that all C, m, t in this section are assumed to be living in the linked time domain. Thus, when we have a $C \in \mathrm{Ctx}_1$, we write C to also mean an element of Ctx_+, with all timestamps equal to its original in Ctx_1 in the second coordinate and with the first coordinate fixed to 0. Likewise, for a $C \in \mathrm{Ctx}_2$, we write C to also mean an element of Ctx_+, with all timestamp equal to its original in Ctx_2 in the second coordinate and with the second coordinate fixed to 1. The same assumption applies for elements in $\mathrm{Mem}_1, \mathrm{Mem}_2, \mathbb{T}_1, \mathbb{T}_2$.

3.1 Definitions

3.1.1 Injection and Deletion

We want to define what it means to *inject* an external $S_1 \subseteq \mathrm{State}_1 \times \mathrm{Tick}_1$ into an assumed $S_2 \subseteq \mathrm{State}_2 \times \mathrm{Tick}_2$. Naturally, we must first define elementwise injection $\rhd$ between $(s_1, \mathsf{tick}_1) \in S_1$ and $(s_2, \mathsf{tick}_2) \in S_2$ and map this over all pairs in $S_1 \times S_2$. What properties must $(s_+, \mathsf{tick}_+) = (s_1, \mathsf{tick}_1) \rhd (s_2, \mathsf{tick}_2)$ satisfy?

Consider the case when we did not assume anything, that is, when $s_2 = ([], \emptyset, 0)$. Then first, we expect that $s_+ \cong s_1$. Second, the tick_+ function under s_+ must preserve the transitions made by tick_2 under s_2. That is, if $(e, s_2) \rightsquigarrow^*_{\mathsf{tick}_2} (e', s'_2)$,

then $(s'_+, \mathsf{tick}'_+) = (s_1, \mathsf{tick}_1) \rhd (s'_2, \mathsf{tick}_2)$ must satisfy $\mathsf{tick}_+ = \mathsf{tick}'_+$ and $(e, s_+) \rightsquigarrow^*_{\mathsf{tick}_+} (e', s'_+)$. This is because we want all transitions after injecting the exported states into the semantics calculated in advance to be valid transitions.

As the first step in defining $\rhd$, we first define the injection operator for contexts, when $C_2\langle C_1\rangle$ "fills in the blank" in C_2 with C_1. The deletion operator $C_2\langle C_1\rangle^{-1}$, which "digs out" C_1 from C_2, is also defined. This definition can be extended

$$C_2\langle C_1\rangle \triangleq \begin{cases} C_1 & C_2 = [] \\ (x,t) :: C\langle C_1\rangle & C_2 = (x,t) :: C \\ (M, C\langle C_1\rangle) :: C'\langle C_1\rangle & C_2 = (M,C) :: C' \end{cases} \quad C_2\langle C_1\rangle^{-1} \triangleq \begin{cases} [] & \hat{\alpha}_+(C_2) = \hat{\alpha}_+(C_1) \\ [] & C_2 = [] \\ (x,t) :: C\langle C_1\rangle^{-1} & C_2 = (x,t) :: C \\ (M, C\langle C_1\rangle^{-1}) :: C'\langle C_1\rangle^{-1} & C_2 = (M,C) :: C' \end{cases}$$

Figure 6: Definition of the injection operator $C_2\langle C_1\rangle$ and the deletion operator $C_2\langle C_1\rangle^{-1}$. Cases are ordered by precedence. For example, we check for $\hat{\alpha}_+(C_2) = \hat{\alpha}_+(C_1)$ first when deleting C_1 from C_2.

to injection and deletion between abstract contexts as well.

Naturally, we expect tick_+ to increment timestamps in $\mathbb{T}_+$ by using tick_1 for timestamps in $2\mathbb{Z} \times \mathbb{T}_1$, and by using tick_2 for timestamps in $(2\mathbb{Z}+1) \times \mathbb{T}_2$. However, the context and memory will contain timestamps originating from both $\mathbb{T}_1$ and $\mathbb{T}_2$. Therefore, we need to define the filter operations $C.1$ and $C.2$ which selects only timestamps from the time domain of interest. This definition can be extended to filter out abstract timestamps as well.

$$C.i \triangleq \begin{cases} [] & C = [] \\ (x,t') :: C'.i & C = (x,t) :: C' \wedge t = (2n-1+i, t') \\ C'.i & C = (x,t) :: C' \wedge t = (2n+i, t') \\ (M, C'.i) :: C''.i & C = (M,C') :: C'' \end{cases} \qquad V.i \triangleq \begin{cases} C.i & V = C \\ \langle e, C.i\rangle & V = \langle e, C\rangle \end{cases}$$

$$m.i \triangleq \bigcup_{(2n-1+i,t) \in \mathsf{dom}(m)} \{t \mapsto m(t).i\}$$

Figure 7: Definition for the filter operations $(i = 1, 2)$.

Note that $m.i$ cannot be well-defined for memories that contain addresses with overlapping second coordinates. That is, in the case when m can be accessed with both $(0, t_1)$ and $(2, t_1)$, $m.1$ is not well-defined. In this case, we leave $m.i$ as undefined. This means that the mapping $m \mapsto m.i$ is a partial function that is defined for only memories with distinct second coordinates. Luckily, in the case that m is derived from injection, all timestamps in $2\mathbb{Z} \times \mathbb{T}_1$ will be of the form $(0, \_)$, and all timestamps in $(2\mathbb{Z}+1) \times \mathbb{T}_2$ will be of the form $(1, \_)$. Thus, $m.i$ will always be defined for our purposes. Of course, $\widehat{m.i}$ is always defined, since the only even first coordinate is 0, and the only odd first coordinate is 1.

Moving on, we extend $\_\langle\_\rangle$ and $\_\langle\_\rangle^{-1}$:

Definition 3.1 (Filling in the Blanks). Let $s_1 = (C_1, m_1, t_1) \in \mathsf{State}_1$ and $r_2 = (V_2, m_2, t_2) \in \mathsf{Result}_2$. Then we define:

$$V_2\langle C_1\rangle \triangleq \begin{cases} C_2\langle C_1\rangle & V_2 = C_2 \\ \langle e, C_2\langle C_1\rangle\rangle & V_2 = \langle e, C_2\rangle \end{cases} \qquad \begin{aligned} m_2\langle C_1\rangle &\triangleq \bigcup_{t \in \mathsf{dom}(m_2)} \{t \mapsto m_2(t)\langle C_1\rangle\} \\ r_2\langle s_1\rangle &\triangleq (V_2\langle C_1\rangle, m_1 \cup m_2\langle C_1\rangle, t_2) \end{aligned}$$

$$V_2\langle C_1\rangle^{-1} \triangleq \begin{cases} C_2\langle C_1\rangle^{-1} & V_2 = C_2 \\ \langle e, C_2\langle C_1\rangle^{-1}\rangle & V_2 = \langle e, C_2\rangle \end{cases} \qquad m_2\langle C_1\rangle^{-1} \triangleq \bigcup_{t \in \mathsf{dom}(m_2)} \left\{t \mapsto m_2(t)\langle C_1\rangle^{-1}\right\}$$

Naturally, these definitions extend to their abstract counterparts.

Note that $r_2\langle s_1\rangle \in \mathsf{Result}_+$. That is, it is time-bounded. Now we only have to define tick_+ which preserves transitions made before injection. The definition for $\rhd$ is:

Definition 3.2 (Elementwise Concrete Injection). Let $s_1 = (C_1, m_1, t_1) \in \mathsf{State}_1$, $\mathsf{tick}_1 \in \mathsf{Tick}_1$, $r_2 = (V_2, m_2, t_2) \in \mathsf{Result}_2$, and $\mathsf{tick}_2 \in \mathsf{Tick}_2$. We define $(s_1, \mathsf{tick}_1) \rhd (r_2, \mathsf{tick}_2) \triangleq (r_2\langle s_1\rangle, \mathsf{tick}_+) \in \mathsf{Result}_+ \times \mathsf{Tick}_+$, when tick_+ is given by:

$$\mathsf{tick}_+(C, m, t, x, v) \triangleq \begin{cases} (2n, \mathsf{tick}_1(C.1, m.1, t_1, x, v.1)) & t = (2n, t_1) \\ (2n+1, \mathsf{tick}_2(C\langle C_1\rangle^{-1}.2, m\langle C_1\rangle^{-1}.2, t_2, x, v\langle C_1\rangle^{-1}.2)) & t = (2n+1, t_2) \\ \hat{\alpha}_+^{-1}(t, \widehat{\mathsf{tick}}_+(\hat{\alpha}_+(C), \hat{\alpha}_+(m), \hat{\alpha}_+(t), x, \hat{\alpha}_+(v))) & m.i = \bot \end{cases}$$

where

$$\widehat{\mathsf{tick}}_+(\widehat{C}, \widehat{m}, \hat{t}, x, \hat{v}) \triangleq \begin{cases} (0, \hat{\alpha}_1(\mathsf{tick}_1)(\widehat{C}.1, \widehat{m}.1, \hat{t}_1, x, \hat{v}.1)) & \hat{t} = (0, \hat{t}_1) \\ (1, \hat{\alpha}_2(\mathsf{tick}_2)(\widehat{C}\langle\hat{\alpha}_1(C_1)\rangle^{-1}.2, \widehat{m}\langle\hat{\alpha}_1(C_1)\rangle^{-1}.2, \hat{t}_2, x, \hat{v}\langle\hat{\alpha}_1(C_1)\rangle^{-1}.2)) & \hat{t} = (1, \hat{t}_2) \end{cases}$$

satisfies $\hat{\alpha}_+ \circ \mathsf{tick}_+ = \widehat{\mathsf{tick}}_+ \circ \hat{\alpha}_+$.

Definition 3.3 (Elementwise Abstract Injection). Let $\hat{s}_1 = (\widehat{C}_1, \widehat{m}_1, \hat{t}_1) \in \widehat{\text{State}}_1$, $\widehat{\text{tick}}_1 \in \widehat{\text{Tick}}_1$, $\hat{r}_2 = (\widehat{V}_2, \widehat{m}_2, \hat{t}_2) \in \widehat{\text{Result}}_2$, and $\widehat{\text{tick}}_2 \in \widehat{\text{Tick}}_2$. We define $(\hat{s}_1, \widehat{\text{tick}}_1)\widehat{\triangleright}(\hat{r}_2, \widehat{\text{tick}}_2) \triangleq (\hat{r}_2\langle\hat{s}_1\rangle, \widehat{\text{tick}}_+) \in \widehat{\text{Result}}_+ \times \widehat{\text{Tick}}_+$, when $\widehat{\text{tick}}_+$ is given by:

$$\widehat{\text{tick}}_+(\widehat{C}, \widehat{m}, \hat{t}, x, \hat{v}) \triangleq \begin{cases} (0, \widehat{\text{tick}}_1(\widehat{C}.1, \widehat{m}.1, \hat{t}_1, x, \hat{v}.1)) & \hat{t} = (0, \hat{t}_1) \\ (1, \widehat{\text{tick}}_2(\widehat{C}\langle\widehat{C}_1\rangle^{-1}.2, \widehat{m}\langle\widehat{C}_1\rangle^{-1}.2, \hat{t}_2, x, \hat{v}\langle\widehat{C}_1\rangle^{-1}.2)) & \hat{t} = (1, \hat{t}_2) \end{cases}$$

Since tick_+ digs out C_1 from the memory and context, timestamps produced by tick_+ after injection will look at only the parts before injection. Thus, it will produce the same timestamps that were produced by tick_2 under S_2. This is why injection into transitions computed in advance are valid as transitions beginning from injected states.

3.1.2 Semantic Linking

$$(s_1, \text{tick}_1) \triangleright (\rho_2, \text{tick}_2) \triangleq \begin{cases} (r_+, \text{tick}_+) & \rho_2 = r_2 \wedge (r_+, \text{tick}_+) = (s_1, \text{tick}_1) \triangleright (r_2, \text{tick}_2) \\ ((e, s_+), \text{tick}_+) & \rho_2 = \ell_2 = (e, s_2) \wedge (s_+, \text{tick}_+) = (s_1, \text{tick}_1) \triangleright (s_2, \text{tick}_2) \end{cases}$$

$$(s_1, \text{tick}_1) \triangleright (\ell_2 \rightsquigarrow_{\text{tick}_2} \rho_2) \triangleq \ell_+ \rightsquigarrow_{\text{tick}_+} \rho_+$$
$$\text{where } (\ell_+, \text{tick}_+) = (s_1, \text{tick}_1) \triangleright (\ell_2, \text{tick}_2) \wedge (\rho_+, \text{tick}_+) = (s_1, \text{tick}_1) \triangleright (\rho_2, \text{tick}_2)$$

Figure 8: Extension of $\triangleright$ to define injection into an element of D_2.

Definition 3.4 (Concrete Injection). Let $S_1 \subseteq \text{State}_1 \times \text{Tick}_1$ and $A_2 \subseteq D_2$. Then we define:

$$S_1 \triangleright A_2 \triangleq \{p \triangleright q | p \in S_1 \wedge q \in A_2\}$$

Definition 3.5 (Abstract Injection). Let $S_1^\# \subseteq \widehat{\text{State}}_1 \times \widehat{\text{Tick}}_1$ and $A_2^\# \subseteq \widehat{D}_2$. Then we define:

$$S_1^\# \triangleright^\# A_2^\# \triangleq \{\hat{p}\widehat{\triangleright}\hat{q} | \hat{p} \in S_1^\# \wedge \hat{q} \in A_2^\#\}$$

Now we need to define the semantic linking operator $\bowtie$. More specifically, we must define $S_1 \bowtie A_2$, when $S_1 \subseteq \text{State}_1 \times \text{Tick}_1$ and $A_2 \subseteq D_2$. Remember that A_2 is the separately computed semantics, and S_1 is what was missing. Thus, we must first inject all $(s_1, \text{tick}_1) \in S_1$ into $(\rho_2, \text{tick}_2), \ell_2 \rightsquigarrow_{\text{tick}_2} \rho_2 \in A_2$. Next, since we have gained new information about the external environment, we must collect more that can be gleaned from S_1. Thus, the definition of semantic linking is as follows:

Definition 3.6 (Concrete Linking). Let $S_1 \subseteq \text{State}_1 \times \text{Tick}_1$ and $A_2 \subseteq D_2$. Then:

$$S_1 \bowtie A_2 \triangleq \text{lfp}(\lambda X.\text{Step}(X) \cup (S_1 \triangleright A_2))$$

Definition 3.7 (Abstract Linking). Let $S_1^\# \subseteq \widehat{\text{State}}_1 \times \widehat{\text{Tick}}_1$ and $A_2^\# \subseteq \widehat{D}_2$. Then:

$$S_1^\# \bowtie^\# A_2^\# \triangleq \text{lfp}(\lambda X^\#.\text{Step}^\#(X^\#) \cup (S_1^\# \triangleright^\# A_2^\#))$$

3.2 Proof Sketches

Lemma 3.1 (Concrete Advance, Operational). For all $(s_1, \text{tick}_1) \in \text{State}_1 \times \text{Tick}_1$, $\ell_2 \in \text{Left}_2$, $\text{tick}_2 \in \text{Tick}_2$, $\rho_2 \in \text{Right}_2$, If we let $(\ell_+, \text{tick}_+) = (s_1, \text{tick}_1) \triangleright (\ell_2, \text{tick}_2)$ and $(\rho_+, \text{tick}_+) = (s_1, \text{tick}_1) \triangleright (\rho_2, \text{tick}_2)$, we have:

$$\ell_2 \rightsquigarrow_{\text{tick}_2} \rho_2 \Rightarrow \ell_+ \rightsquigarrow_{\text{tick}_+} \rho_+$$

Thus $S_1 \triangleright [\![e]\!]S_2 \subseteq [\![e]\!](S_1 \triangleright S_2)$.

Sketch. By induction on $\rightsquigarrow_{\text{tick}_2}$. We use the fact that $m[t \mapsto v]\langle C\rangle = m\langle C\rangle[t \mapsto v\langle C\rangle]$. □

Lemma 3.2 (Concrete Advance). Let $S_1 \subseteq \text{State}_1 \times \text{Tick}_1$ and $S_2 \subseteq \text{State}_2 \times \text{Tick}_2$. Then:

$$[\![e]\!](S_1 \triangleright S_2) = S_1 \bowtie [\![e]\!]S_2$$

Sketch. Straightforward from the previous lemma. □

Lemma 3.3 (Abstract Advance, Operational). For all $(\hat{s}_1, \widehat{\mathsf{tick}}_1) \in \widehat{\mathsf{State}}_1 \times \widehat{\mathsf{Tick}}_1$, $\hat{\ell}_2 \in \widehat{\mathsf{Left}}_2$, $\widehat{\mathsf{tick}}_2 \in \widehat{\mathsf{Tick}}_2$, $\hat{\rho}_2 \in \widehat{\mathsf{Right}}_2$, If we let $(\hat{\ell}_+, \widehat{\mathsf{tick}}_+) = (\hat{s}_1, \widehat{\mathsf{tick}}_1)\widehat{\triangleright}(\hat{\ell}_2, \widehat{\mathsf{tick}}_2)$ and $(\hat{\rho}_+, \widehat{\mathsf{tick}}_+) = (\hat{s}_1, \widehat{\mathsf{tick}}_1)\widehat{\triangleright}(\hat{\rho}_2, \widehat{\mathsf{tick}}_2)$, we have:

$$\hat{\ell}_2 \mathrel{\widehat{\rightsquigarrow}}_{\widehat{\mathsf{tick}}_2} \hat{\rho}_2 \Rightarrow \hat{\ell}_+ \mathrel{\widehat{\rightsquigarrow}}_{\widehat{\mathsf{tick}}_+} \hat{\rho}_+$$

Thus $S_1^\# \triangleright^\# [\![e]\!]^\# S_2^\# \subseteq [\![e]\!]^\# (S_1^\# \triangleright^\# S_2^\#)$.

Sketch. By induction on $\mathrel{\widehat{\rightsquigarrow}}_{\widehat{\mathsf{tick}}_2}$. We use the fact that $\widehat{m}[\hat{t} \mathrel{\widehat{\mapsto}} \hat{v}]\langle\widehat{C}\rangle = \widehat{m}\langle\widehat{C}\rangle[\hat{t} \mathrel{\widehat{\mapsto}} \hat{v}\langle\widehat{C}\rangle]$. □

Lemma 3.4 (Abstract Advance). Let $S_1^\# \subseteq \widehat{\mathsf{State}}_1 \times \widehat{\mathsf{Tick}}_1$ and $S_2^\# \subseteq \widehat{\mathsf{State}}_2 \times \widehat{\mathsf{Tick}}_2$. Then:

$$[\![e]\!]^\# (S_1^\# \triangleright^\# S_2^\#) = S_1^\# \between^\# [\![e]\!]^\# S_2^\#$$

Sketch. Straightforward from the previous lemma. □