모듈별 프로그램 따로 분석

이준협

2023년 8월 4일

ROPAS Show & Tell

Problem Statement

- Given two program segments e_1 and e_2,
- Want to derive a sound approximation of $[\![e_1 ! e_2]\!]$ utilizing information obtained from *separately* analyzing e_1 and e_2

Abstract Syntax

$$
\begin{array}{lcll}
x & \in & \mathsf{ExprVar} & \\
M & \in & \mathsf{ModVar} & \\
e & \in & \mathsf{Expr} & \\
e & ::= & x & \textit{identifier, expression} \\
 & \mid & \lambda x.e & \textit{function} \\
 & \mid & e\ \ e & \textit{application} \\
 & \mid & e!e & \textit{linked expression} \\
 & \mid & \varepsilon & \textit{empty module} \\
 & \mid & M & \textit{identifier, module} \\
 & \mid & \texttt{let}\ x\ \ e\ \ e & \textit{let-binding, expression} \\
 & \mid & \texttt{let}\ M\ \ e\ \ e & \textit{let-binding, module}
\end{array}
$$

Semantic Domains

$$
\begin{array}{rccl}
\text{Time} & t & \in & \mathbb{T} \\
\text{Environment/Context} & C & \in & \mathsf{Ctx}(\mathbb{T}) \\
\text{Value(Expr)} & v & \in & \mathsf{Val}(\mathbb{T}) \triangleq \mathsf{Expr} \times \mathsf{Ctx}(\mathbb{T}) \\
\text{Value(Expr/Mod)} & V & \in & \mathsf{Val}(\mathbb{T}) + \mathsf{Ctx}(\mathbb{T}) \\
\text{Memory} & m & \in & \mathsf{Mem}(\mathbb{T}) \triangleq \mathbb{T} \xrightarrow{\text{fin}} \mathsf{Val}(\mathbb{T}) \\
\text{State} & s & \in & \mathsf{State}(\mathbb{T}) \triangleq \mathsf{Ctx}(\mathbb{T}) \times \mathsf{Mem}(\mathbb{T}) \times \mathbb{T} \\
\text{Result} & r & \in & \mathsf{Result}(\mathbb{T}) \triangleq (\mathsf{Val}(\mathbb{T}) + \mathsf{Ctx}(\mathbb{T})) \times \mathsf{Mem}(\mathbb{T}) \times \mathbb{T} \\
\text{Context} & C & \to & [] \\
 & & \mid & (x, t) :: C \\
 & & \mid & (M, C) :: C \\
\text{Value(Expr)} & v & \to & \langle \lambda x.e, C \rangle
\end{array}
$$

Time?

Concrete Time

$(\mathbb{T}, \leq, \text{tick})$ is a *concrete time* when

1. $(\mathbb{T}, \leq)$ is a total order.
2. $\text{tick} : \text{Ctx}(\mathbb{T}) \to \text{Mem}(\mathbb{T}) \to \mathbb{T} \to \text{ExprVar} \to \text{Val}(\mathbb{T}) \to \mathbb{T}$ satisfies:

$$\forall t \in \mathbb{T} : t < \text{tick} \_\ \_\ t \_\ \_$$

Big-Step Evaluation Relation(Excerpt)

$$\boxed{(e, C, m, t) \Downarrow (V, m', t')}$$

$$[\textsc{ExprVar}]\ \frac{t_x = \mathsf{addr}(C, x) \qquad v = m(t_x)}{(x, C, m, t) \Downarrow (v, m, t)}$$

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

$$[\textsc{Empty}]\ \frac{}{(\varepsilon, C, m, t) \Downarrow (C, m, t)}$$

Type of the Collecting Semantics: Take 1

$$[\![e]\!](s) \in ?$$

- Collecting semantics of the expression e under initial state s.
- Collect *all* configurations in the proof tree starting from (e, s).
- Why? To *inject* the exported context into the incomplete proof tree.
- Need to remember all (e, s) the big-step interpreter *saw*.
- Need to define what these “reachable configurations” are.

Big-Step Reachability Relation(Excerpt)

$$\boxed{(e, C, m, t) \rightsquigarrow (e', C', m', t')}$$

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

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

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

Type of the Collecting Semantics

$$[\![e]\!](s) \in (\mathsf{Expr} \times \mathsf{State}(\mathbb{T})) \to \wp(\mathsf{Result}(\mathbb{T}))$$

- Remember: All configurations (e', s') the interpreter “saw”
- Remember: All results r' the interpreter returned
- Collecting semantics: A cache recording *reached configurations* with the *results* they return.
- Collecting semantics: “History” of the interpreter

Definition of the Transfer Function: Take 1

- Given a cache a, need to simulate one step of the interpreter.
- "One-step": Related by $\rightsquigarrow$.
- "Given a cache": The interpreter is constrained by its history a.
- "Constrained": All assumptions in $\rightsquigarrow$ need to be *in* a.

One step of Transfer

- Define $\Downarrow_a$ and $\rightsquigarrow_a$ by replacing all premises $(e, s) \Downarrow r$ by $r \in a(e, s)$ in $\Downarrow$ and $\rightsquigarrow$.
- Define the step function that does what the interpreter will do with input (e, s) under a.

$$\mathrm{step}(a)(e, s) \triangleq [(e, s) \mapsto \{r | (e, s) \Downarrow_a r\}] \cup \bigcup_{(e,s) \rightsquigarrow_a (e',s')} [(e', s') \mapsto \varnothing]$$

Definition of the Transfer Function & Collecting Semantics

We define the transfer function Step by:

$$\mathsf{Step}(a) \triangleq \bigcup_{(e,s)\in\mathsf{dom}(a)} \mathsf{step}(a)(e, s)$$

Then:

Definition (Collecting Semantics)

$$[\![e]\!](s) \triangleq \mathsf{lfp}(\lambda a.\mathsf{Step}(a) \cup [(e, s) \mapsto \emptyset])$$

Concrete Linking

- Need to link *separately* computed semantics in $\mathbb{T}_1$ and $\mathbb{T}_2$.
- Why? The abstract must approximate the concrete *separately*.
- In other words, construct "tick_+" approximated by "$\text{tick}_+^\#$"

Requirements for tick_+

The tick_+ in the linked time domain must:

1. Increment $(0_1, 0_2)$ up to $(t_1, 0_2)$, when $t_1 \in \mathbb{T}_1$ is the final timestamp of $[\![e_1]\!](\_, \_, 0_1)$.
2. Increment $(t_1, 0_2)$ up to (t_1, t_2), when $t_2 \in \mathbb{T}_2$ is the largest timestamp that can be computed *without* knowing about what e_1 exports to e_2.

How to Construct tick_{+}

- The second timestamp starts to tick under $(C_1, m_1, (t_1, 0_2))$, when $(e_1, s) \Downarrow (C_1, m_1, t_1)$.
- The second timestamp was originally ticked under $([], \varnothing, 0_2)$.
- Need to *dig out* C_1 from the stack and *filter out* m_1.

How to Dig Out C: Injection

- Need to determine *what part* of the context is exported.
- Since there are no signatures, the exported context is automatically included in module bindings.
- The definition of the injection operator $C_1\langle C_2\rangle$ is mutually recursive with the injection that maps over all module bindings $C_1[C_2]$.
- Note: $+\!\!+$ is the list append operator.

$$C_1[C_2] \triangleq \begin{cases} [] & C_2 = [] \\ (x,t) :: C_1[C'] & C_2 = (x,t) :: C' \\ (M, C_1\langle C'\rangle) :: C_1[C''] & C_2 = (M, C') :: C'' \end{cases} \qquad C_1\langle C_2\rangle \triangleq C_1[C_2] +\!\!+ C_1$$

How to Dig Out C: Deletion

- The deletion operators are defined as inverse operators of $+\!\!+$, $C_1[C_2]$, $C_1\langle C_2\rangle$.
- The deletion operators satisfy $(C_2 +\!\!+ C_1)\overline{+\!\!+}C_1 = C_2$, $C_1\overline{[C_1[C_2]]} = C_2$, and $C_1\overline{\langle C_1\langle C_2\rangle\rangle} = C_2$.

$$C_2\overline{+\!\!+}C_1 \triangleq \begin{cases} C_2'\overline{+\!\!+}C_1' & (C_1, C_2) = (C_1' +\!\!+ [(x,t)], C_2'[(x,t)]) \\ C_2'\overline{+\!\!+}C_1' & (C_1, C_2) = (C_1' +\!\!+ [(M,C)], C_2' +\!\!+ [(M,C)]) \\ C_2 & \text{otherwise} \end{cases}$$

$$C_1\overline{[C_2]} \triangleq \begin{cases} [] & C_2 = [] \\ (x,t) :: C_1\overline{[C']} & C_2 = (x,t) :: C' \\ (M, C_1\overline{\langle C'\rangle}) :: C_1\overline{[C'']} & C_2 = (M, C') :: C'' \end{cases} \qquad C_1\overline{\langle C_2\rangle} \triangleq C_1\overline{[C_2\overline{+\!\!+}C_1]}$$

How to Filter Out m

All timestamps incremented after $(t_1, 0_2)$ will have $t.1 = t_1$.

$$\mathsf{filter}_1(C) \triangleq \begin{cases} [] & C = [] \\ (x, t.1) :: \mathsf{filter}_1(C') & C = (x,t) :: C' \wedge t.1 \neq t_1 \\ \mathsf{filter}_1(C') & C = (x,t) :: C' \wedge t.1 = t_1 \\ (M, \mathsf{filter}_1(C')) :: \mathsf{filter}_1(C'') & C = (M, C') :: C'' \end{cases}$$

$$\mathsf{filter}_2(C) \triangleq \begin{cases} [] & C = [] \\ (x, t.2) :: \mathsf{filter}_2(C') & C = (x,t) :: C' \wedge t.1 = t_1 \\ \mathsf{filter}_2(C') & C = (x,t) :: C' \wedge t.1 \neq t_1 \\ (M, \mathsf{filter}_2(C')) :: \mathsf{filter}_2(C'') & C = (M, C') :: C'' \end{cases}$$

Definition of tick_+

$$\mathsf{tick}_+(C,m,t,x,v) \triangleq \begin{cases} (\mathsf{tick}_1\ \mathsf{filter}_1(C,m,t,x,v), 0_2) & (t.1 \neq t_1) \\ (t_1, \mathsf{tick}_2\ \mathsf{filter}_2(C_1\overline{\langle C,m,t,x,v\rangle})) & (t.1 = t_1) \end{cases}$$

- The timestamps that tick_+ increments live in

$$\mathbb{T}_1 \otimes \mathbb{T}_2 \triangleq \mathbb{T}_1 \times \{0_2\} \cup \{t_1\} \times \mathbb{T}_2$$

- From now on, assume that all $(C,m,t) \in \mathsf{State}(\mathbb{T})$ satisfy the invariant that all timestamps in C and m are smaller than t.
- Then $(\mathbb{T}_1 \otimes \mathbb{T}_2, \leq_+, \mathsf{tick}_+)$, when $\leq_+$ is the lexicographic order, is a concrete time that *preserves timestamps*.

Preservation of Timestamps

- The separately ticked timestamps must agree with the timestamps ticked with tick_+ under the *injected* context.
- First, define injection of $s \in \mathsf{State}(\mathbb{T}_1)$ as:

$$s \triangleright m_2 \triangleq \lambda t. \begin{cases} m_1(t.1) & t.1 \in \mathsf{dom}(m_1) \wedge t.2 = 0_2 \\ C_1\langle m_2\rangle(t.2) & t.1 = t_1 \wedge t.2 \in \mathsf{dom}(m_2) \end{cases}$$

and

$$s \triangleright r \triangleq (C_1\langle V_2\rangle, s \triangleright m_2, t_2)$$

and

$$s \triangleright a \triangleq \bigcup_{(e,s')\in\mathsf{dom}(a)} [(e, s \triangleright s') \mapsto \{s \triangleright r | r \in a(e, s')\}]$$

Preservation of Timestamps

Lemma (Injection preserves timestamps under linked time)

$$\forall s \in \mathit{State}(\mathbb{T}_1), s' \in \mathit{State}(\mathbb{T}_2) \,:\, s \rhd [\![e]\!](s') \sqsubseteq [\![e]\!](s \rhd s')$$

Concrete Linking

Definition (Auxiliary operators for concrete linking)

$$\mathsf{Exp}\ e_1\ s \triangleq [\![e_1]\!](s)(e_1, s) \qquad \text{(Exported under } s\text{)}$$

$$\mathsf{L}\ E\ e_2 \triangleq \bigcup_{s' \in E} [\![e_2]\!](s' \rhd 0_2) \qquad \text{(Reached under } E\text{)}$$

$$\mathsf{F}\ E\ e_2 \triangleq \bigcup_{s' \in E} [\![e_2]\!](s' \rhd 0_2)(e_2, s' \rhd 0_2) \qquad \text{(Final results under } E\text{)}$$

Definition (Concrete linking operator)

$$\mathsf{Link}\ e_1\ e_2\ s \triangleq [\![e_1]\!](s) \cup \mathsf{L}\ (\mathsf{Exp}\ e_1\ s)\ e_2 \cup [(e_1!e_2, s) \mapsto \mathsf{F}\ (\mathsf{Exp}\ e_1\ s)\ e_2]$$

When all timestamps $t \in \mathbb{T}_1$ are lifted to $(t, 0_2)$.

Abstract Semantics : What Configurations Look Like

$$
\begin{array}{lcl}
t^\# & \in & \mathbb{T}^\# \\
v^\# & \in & \mathsf{Val}(\mathbb{T}^\#) \\
C^\# & \in & \mathsf{Ctx}(\mathbb{T}^\#) \\
V^\# & \in & \mathsf{Val}(\mathbb{T}^\#) + \mathsf{Ctx}(\mathbb{T}^\#) \\
m^\# & \in & \mathsf{Mem}^\#(\mathbb{T}^\#) \triangleq \mathbb{T}^\# \xrightarrow{\mathsf{fin}} \wp(\mathsf{Val}(\mathbb{T}^\#)) \\
s^\# & \in & \mathsf{State}^\#(\mathbb{T}^\#) \triangleq \mathsf{Ctx}(\mathbb{T}^\#) \times \mathsf{Mem}^\#(\mathbb{T}^\#) \times \mathbb{T}^\# \\
r^\# & \in & \mathsf{Result}^\#(\mathbb{T}^\#) \triangleq (\mathsf{Val}(\mathbb{T}^\#) + \mathsf{Ctx}(\mathbb{T}^\#)) \times \mathsf{Mem}^\#(\mathbb{T}^\#) \times \mathbb{T}^\#
\end{array}
$$

Time?

Definition (Abstract time)

$(\mathbb{T}^\#, \mathrm{tick}^\#)$ is an *abstract time* when
$\mathrm{tick}^\# \in \mathrm{Ctx}(\mathbb{T}^\#) \to \mathrm{Mem}^\#(\mathbb{T}^\#) \to \mathbb{T}^\# \to \mathrm{ExprVar} \to \mathrm{Val}(\mathbb{T}^\#) \to \mathbb{T}^\#$
is the policy for advancing the timestamp.

Big-Step Evaluation Relation

$$[\textsc{ExprVar}]\ \frac{{t_x}^{\#} = \mathsf{addr}(C^{\#}, x) \qquad v^{\#} \in m^{\#}({t_x}^{\#})}{(x, C^{\#}, m^{\#}, t^{\#}) \Downarrow^{\#} (v^{\#}, m^{\#}, t^{\#})}$$

Big-Step Evaluation Relation

$$[\textsc{ExprVar}]\ \frac{{t_x}^{\#} = \mathsf{addr}(C^{\#}, x) \qquad v^{\#} \in m^{\#}({t_x}^{\#})}{(x, C^{\#}, m^{\#}, t^{\#}) \Downarrow^{\#} (v^{\#}, m^{\#}, t^{\#})}$$

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

Abstract Semantics

Abstract Semantics

The semantics for an expression e under configuration $s^\# \in \mathsf{State}^\#(\mathbb{T}^\#)$ is an element in $(\mathsf{Expr} \times \mathsf{State}^\#(\mathbb{T}^\#)) \to (\wp(\mathsf{Result}^\#(\mathbb{T}^\#)))_\perp$ defined as:

$$[\![e]\!]^\#(s^\#) \triangleq \bigsqcup_{(e,s^\#) \rightsquigarrow^{\#*} (e',s'^\#)} [(e', s'^\#) \mapsto \{r^\# | (e', s'^\#) \Downarrow^\# r^\#\}]$$

$$[\![e]\!]^\#(s^\#) = \mathsf{lfp}(\lambda f^\#.F^\#([(e, s^\#) \mapsto \varnothing] \sqcup f^\#))$$

Transfer Function

Definition (Transfer function)

Given an element $f^\#$ of $(\mathsf{Expr} \times \mathsf{State}^\#(\mathbb{T}^\#)) \to (\wp(\mathsf{Result}^\#(\mathbb{T}^\#)))_\perp$,

- Define $\Downarrow^\#_{f^\#}$ and $\rightsquigarrow^\#_{f^\#}$ by replacing all assumptions of the form $s^\# \Downarrow^\# r^\#$ to $r^\# \in f^\#(s^\#)$ in $\Downarrow^\#$ and $\rightsquigarrow^\#$.

We define the transfer function $F^\#$ by:

$$F^\#(f^\#) \triangleq f^\# \sqcup \bigsqcup_{\substack{(e, s^\#) \in \mathsf{dom}(f^\#) \\ (e, s^\#) \rightsquigarrow^\#_{f^\#} (e', s'^\#)}} [(e', s'^\#) \mapsto \{r^\# | (e', s'^\#) \Downarrow^\#_{f^\#} r^\#\}]$$

Soundness Given α

Definition (α-soundness between results)

- Let $(V, m, t) \in \mathsf{Result}(\mathbb{T})$ and $(V^\#, m^\#, t^\#) \in \mathsf{Result}^\#(\mathbb{T}^\#)$.
- Let $\alpha : \mathbb{T} \to \mathbb{T}^\#$, and extend α to a function in $\mathsf{Ctx}(\mathbb{T}) \to \mathsf{Ctx}(\mathbb{T}^\#)$ by mapping α over all timestamps.
- Extend α to a function in $(\mathsf{Ctx}(\mathbb{T}) + \mathsf{Val}(\mathbb{T})) \to (\mathsf{Ctx}(\mathbb{T}^\#) + \mathsf{Val}(\mathbb{T}^\#))$
- Extend α to a function in $\mathsf{Mem}(\mathbb{T}) \to \mathsf{Mem}^\#(\mathbb{T}^\#)$ by defining

$$\alpha(m) \triangleq \bigsqcup_{t \in \mathsf{dom}(m)} [\alpha(t) \mapsto \{\alpha(m(t))\}]$$

We say that $(V^\#, m^\#, t^\#)$ is an *α-sound approximation* of (V, m, t) when $\alpha(V) = V^\#$, $\alpha(m) \sqsubseteq m^\#$, and $\alpha(t) = t^\#$.

Soundness

Definition (Soundness between semantics)

- Let $f \in (\mathrm{Expr} \times \mathrm{State}(\mathbb{T})) \to (\wp(\mathrm{Result}(\mathbb{T})))_\perp$ and $f^\# \in (\mathrm{Expr} \times \mathrm{State}^\#(\mathbb{T}^\#)) \to (\wp(\mathrm{Result}^\#(\mathbb{T}^\#)))_\perp$.

We say that $f^\#$ is a sound approximation of f if:

$$\begin{aligned} \forall e \in \mathrm{Expr}, s \in \mathrm{State}(\mathbb{T}), r \in \mathrm{Result}(\mathbb{T}) \ : \\ r \in f(e,s) \Rightarrow \\ \exists \alpha, \alpha', s^\#, r^\# \ : \ \alpha(s) \sqsubseteq s^\# \wedge \alpha'(r) \sqsubseteq r^\# \in f^\#(e, s^\#) \end{aligned}$$

Why?

Preservation of soundness

- Let $s \in \mathsf{State}(\mathbb{T})$ and $s^\# \in \mathsf{State}^\#(\mathbb{T}^\#)$.
- Let all timestamps in the C and m component of s be strictly less than the t component.
- Let $s^\#$ be an α-sound approximation of s for some α.

Then for all e, $[\![e]\!]^\#(s^\#)$ is a sound approximation of $[\![e]\!](s)$.

Abstract Linking

Define an *injection* operator that, given $s^\# \in \mathsf{State}^\#(\mathbb{T}^\#), r^\# \in \mathsf{Result}^\#(\mathbb{T}'^\#)$, gives $s^\# \rhd r^\# \in \mathsf{Result}^\#((\mathbb{T}^\# + \mathbb{T}'^\#))$, that satisfies:

1. $\alpha(s) \sqsubseteq s^\# \Rightarrow \exists \alpha' : \alpha'(s) \sqsubseteq (s^\# \rhd \emptyset)$.
2. $\exists \mathsf{tick}_+^\#$ such that $(\mathbb{T}^\# + \mathbb{T}'^\#, \mathsf{tick}_+^\#)$ is an abstract time, and

$$s^\# \rhd [\![e]\!]^\#(s'^\#) \sqsubseteq [\![e]\!]^\#(s^\# \rhd s'^\#)$$

Abstract Linking

Recall:

$$[\![e_1!e_2]\!](s) = [\![e_1]\!](s) \sqcup \bigsqcup_{s' \in [\![e_1]\!](s)(e_1,s)} ([\![e_2]\!](s') \sqcup [(e_1!e_2, s) \mapsto [\![e_2]\!](s')(e_2, s')])$$

Abstract Linking

Recall:

$$[\![e_1!e_2]\!](s) = [\![e_1]\!](s) \sqcup \bigsqcup_{s' \in [\![e_1]\!](s)(e_1,s)} \left([\![e_2]\!](s') \sqcup [(e_1!e_2, s) \mapsto [\![e_2]\!](s')(e_2, s')]\right)$$

1. Given a sound approximation $f^\#$ of $[\![e_1]\!](s)$ under $\mathbb{T}^\#$, extract a set containing α-sound approximations of all exported configurations s'.

Abstract Linking

Recall:

$$[\![e_1!e_2]\!](s) = [\![e_1]\!](s) \sqcup \bigsqcup_{s' \in [\![e_1]\!](s)(e_1,s)} ([\![e_2]\!](s') \sqcup [(e_1!e_2, s) \mapsto [\![e_2]\!](s')(e_2, s')])$$

1. Given a sound approximation $f^\#$ of $[\![e_1]\!](s)$ under $\mathbb{T}^\#$, extract a set containing α-sound approximations of all exported configurations s'.
2. Inject the exported context onto the separately analyzed results $[\![e_2]\!]^\#(\emptyset)$, then perform the fixpoint computation starting from there to obtain $[\![e_2]\!]^\#(s'^\#)$ which is a sound approximation of $[\![e_2]\!](s')$.

Why Can We Compute $[\![e]\!]^{\#}$?

Theorem (Finiteness of time implies finiteness of abstraction)

If $\mathbb{T}^{\#}$ is finite,

$$\forall e, s^{\#} \; : \; |[\![e]\!]^{\#}(s^{\#})| < \infty$$

감사합니다