Lightweight Polymorphic Effects - Proofs

Lukas Rytz, Martin Odersky, and Philipp Haller

EPFL, Switzerland,
`{first.last}@epfl.ch`

EPFL-REPORT-173820

Abstract. This technical report presents the full proofs for the type preservation and effect soundness theorems of the type system presented in the article "Lightweight Polymorphic Effects".

1 Introduction

We first repeat the lemmas and theorems introduced in the paper [2].

Lemma 1. *Monotonicity.*
For every effect domain $\mathcal{D}$ and every syntactic form $\textsc{Trm}$,

1. *if* $\forall\ e_i \in e, d_i \in d\ .\ e_i \sqsubseteq d_i$, *then* $\mathrm{eff}_{\mathcal{D}}(\textsc{Trm}, \overline{e}) \sqsubseteq \mathrm{eff}_{\mathcal{D}}(\textsc{Trm}, \overline{d})$
2. $\mathrm{eff}_{\mathcal{D}}(\textsc{Trm}, e_1, \ldots, e_{i1} \sqcup e_{i2}, \ldots, e_n) \sqsubseteq \mathrm{eff}_{\mathcal{D}}(\textsc{Trm}, e_1, \ldots, e_{i1}, \ldots, e_n) \sqcup e_{i2}$

Lemma 2. *Consistency.*

- *Let $S = \mathrm{dynEff}(\textsc{Trm}, \overline{S})$ be the set of dynamic effects that occur when evaluating a term t of the form* $\textsc{Trm}$. *The list $\overline{S}$ contains an effect set for every subterm of t.*
- *Let $\Gamma; f$ be an environment and $\overline{e}$ be a list of static effects such that every effect set in $\overline{S}$ is approximated by $S_i \preceq e_i \sqcup \mathrm{latent}(\Gamma(f))$.*
- *Then the static effect $e = \mathrm{eff}(\textsc{Trm}, \overline{e})$ is a conservative approximation of the effects in S, i.e., $S \preceq e \sqcup \mathrm{latent}(\Gamma(f))$.*

As noted in the paper, this lemma has to be verified for the $\mathit{dynEff}_{\mathcal{D}}$ and $\mathit{eff}_{\mathcal{D}}$ functions of every effect domain $\mathcal{D}$. We show that it holds for the domain of exceptions $\mathcal{E}$, and we then show that it holds for the general functions *dynEff* and *eff* which act on all effect domains.

Lemma 3. *Preservation under substitution for monomorphic abstractions.*
If $\Gamma, x : T_1; f \vdash t : T \ !\ e_l$, $f \neq x$ and $\Gamma; g \vdash v : T_2 \ !\ \bot$ with $T_2 <: T_1$, then $\Gamma; f \vdash t[v \backslash x] : T' \ !\ e_l'$ such that $T' <: T$ and $e_l' \sqsubseteq e_l$.

Lemma 4. *Preservation under substitution for polymorphic abstractions.*
If $\Gamma, x : T_1; x \vdash t : T \ !\ e_l$ and $\Gamma; g \vdash v : T_2 \ !\ \bot$ with $T_2 <: T_1$, then $\Gamma; \epsilon \vdash t[v \backslash x] : T' \ !\ e_l'$ such that $T' <: T$ and $e_l' \sqsubseteq e_l \sqcup \mathrm{latent}(T_2)$.

The substitution lemmas are used in the proofs of both the preservation and the soundness theorem.

Theorem 1. *Preservation.*
If $\Gamma; f \vdash t : T \; ! \; e$ *is a valid typing statement for term* t *and the term evaluates as* $t \Downarrow \langle r, S \rangle$*, then there is valid a typing statement* $\Gamma; f \vdash r : T' \; ! \; e'$ *for* r *with* $T' <: T$*.*

Theorem 2. *Effect soundness.*
If $\Gamma; f \vdash t : T \; ! \; e$ *and* $t \Downarrow \langle r, S \rangle$*, then* $S \preceq e \sqcup \mathrm{latent}(\Gamma(f))$*.*

2 Consistency of *eff* and *dynEff*, monotonicity of *eff*

2.1 Consistency of *eff* and *dynEff*

Proof (Lemma 2 for the domain of exceptions $\mathcal{E}$*).* By case-analysis on the syntactic terms Trm:

- case $\textsc{Throw}(p)$:
  $S = \mathit{dynEff}_{\mathcal{E}}(\textsc{Throw}(p)) = \mathsf{throws}(p)$
  $e = \mathit{eff}_{\mathcal{E}}(\textsc{Throw}(p)) = \mathsf{throws}(p)$
  therefore $S \preceq e$.
- case Catch:
  $S = \mathit{dynEff}_{\mathcal{E}}(\textsc{Catch}, S_1, S_2) = (S_1 \setminus \{throws(p_i) \mid p_i \in \overline{p}\}) \cup S_2$
  $e = \mathit{eff}_{\mathcal{E}}(\textsc{Catch}, e_1, e_2) = \mathsf{throws}((\overline{q} \setminus \overline{p}) \cup \overline{s})$,
  where $e_1 = \mathsf{throws}(\overline{q})$, $e_2 = \mathsf{throws}(\overline{s})$
  For every effect $e_S \in S$, there are two cases:
  - case $e_S \in (S_1 \setminus \{throws(p_i) \mid p_i \in \overline{p}\})$: For every $e_S = \mathsf{throws}(q)$, we have $q \notin \overline{p}$. By hypothesis $\forall e_{S1} \in S_1 \,.\, e_{S1} \sqsubseteq \mathsf{throws}(\overline{q}) \sqcup \mathit{latent}(\Gamma(f))$, so either $q \in \overline{q}$ or $e_{S1} \sqsubseteq \mathit{latent}(\Gamma(f))$. We obtain the result $e_S = \mathsf{throws}(q) \sqsubseteq \mathsf{throws}(\overline{q} \setminus \overline{p}) \sqcup \mathit{latent}(\Gamma(f))$.
  - case $e_S \in S_2$: by hypothesis, $e_S \sqsubseteq \mathsf{throws}(\overline{s}) \sqcup \mathit{latent}(\Gamma(f))$.

  Given $\forall e_S \,.\, e_S \sqsubseteq \mathsf{throws}(\overline{q} \setminus \overline{p}) \;\vee\; e_S \sqsubseteq \mathsf{throws}(\overline{s}) \;\vee\; e_S \sqsubseteq \mathit{latent}(\Gamma(f))$, we conclude $e_S \sqsubseteq \mathsf{throws}((\overline{q} \setminus \overline{p}) \cup \overline{s}) \sqcup \mathit{latent}(\Gamma(f))$.
- Remaining cases (including Try):
  $S = \mathit{dynEff}_{\mathcal{E}}(\textsc{Trm}, \overline{S}) = \bigcup S_i$
  $e = \mathit{eff}_{\mathcal{E}}(\textsc{Trm}, \overline{e}) = \bigsqcup e_i$
  $\forall e_{Si} \in S_i \,.\, e_{Si} \sqsubseteq e_i \sqcup \mathit{latent}(\Gamma(f))$ by hypothesis of the lemma. By the lattice properties of $\sqcup$ and $\sqsubseteq$, we conclude $\forall e_S \in S \,.\, e_S \sqsubseteq e \sqcup \mathit{latent}(\Gamma(f))$.

$\square$

Proof (Lemma 2 for the multi-domain functions dynEff *and* eff*).* Immediate by using the effect-domain specific versions of Lemma 2 element-wise on the effects in S and e. $\square$

2.2 Monotonicity of *eff*

Proof (Lemma 1, Part 1., for the domain of exceptions $\mathcal{E}$*).* By case-analysis on the syntactic terms Trm:

– case $\textsc{Catch}$:
$\mathit{eff}_{\mathcal{E}}(\textsc{Catch}(\overline{p}), e_1, e_2) = \mathtt{throws}((\overline{q_e} \setminus \overline{p}) \cup \overline{s_e})$,
where $e_1 = \mathtt{throws}(\overline{q_e})$ and $e_2 = \mathtt{throws}(\overline{s_e})$
$\mathit{eff}_{\mathcal{E}}(\textsc{Catch}(\overline{p}), d_1, d_2) = \mathtt{throws}((\overline{q_d} \setminus \overline{p}) \cup \overline{s_d})$,
where $d_1 = \mathtt{throws}(\overline{q_d})$ and $d_2 = \mathtt{throws}(\overline{s_d})$
Since $e_1 \sqsubseteq d_1, e_2 \sqsubseteq d_2$, we have $\overline{q_e} \subseteq \overline{q_d}$ and $\overline{s_e} \subseteq \overline{s_d}$. By the properties of set operations, $((\overline{q_e} \setminus \overline{p}) \cup \overline{s_e}) \subseteq ((\overline{q_d} \setminus \overline{p}) \cup \overline{s_d})$.
Therefore, $\mathit{eff}_{\mathcal{E}}(\textsc{Catch}(\overline{p}), e_1, e_2) \sqsubseteq \mathit{eff}_{\mathcal{E}}(\textsc{Catch}(\overline{p}), d_1, d_2)$.
– Remaining cases (including $\textsc{Throw}(p)$, $\textsc{Try}$):
We have $\mathit{eff}_{\mathcal{E}}(\textsc{Trm}, \overline{e}) = \bigsqcup e_i$ and $\mathit{eff}_{\mathcal{E}}(\textsc{Trm}, \overline{d}) = \bigsqcup d_i$. Since $e_i \sqsubseteq d_i$, the conclusion is immediate.

□

Proof (Lemma 1, Part 2., for the domain of exceptions $\mathcal{E}$). By case-analysis on the syntactic terms $\textsc{Trm}$:

– case $\textsc{Catch}$ where $e_1 = e_{11} \sqcup e_{12}$:
$\mathit{eff}_{\mathcal{E}}(\textsc{Catch}(\overline{p}), e_{11} \sqcup e_{12}, e_2) = \mathtt{throws}(((\overline{q_{e1}} \cup \overline{q_{e2}}) \setminus \overline{p}) \cup \overline{s_e})$
$= \mathtt{throws}((\overline{q_{e1}} \setminus \overline{p}) \cup \overline{s_e} \cup (\overline{q_{e2}} \setminus \overline{p}))$,
where $e_{11} = \mathtt{throws}(\overline{q_{e1}})$, $e_{12} = \mathtt{throws}(\overline{q_{e2}})$ and $e_2 = \mathtt{throws}(\overline{s_e})$
$\mathit{eff}_{\mathcal{E}}(\textsc{Catch}(\overline{p}), e_{11}, e_2) \sqcup e_{12} = \mathtt{throws}((\overline{q_{e1}} \setminus \overline{p}) \cup \overline{s_e}) \sqcup \mathtt{throws}(\overline{q_{e2}})$.
The conclusion is straightforward.
– case $\textsc{Catch}$ where $e_2 = e_{21} \sqcup e_{22}$: Similar as above.
– Remaining cases (including $\textsc{Throw}(p)$, $\textsc{Try}$):
We have $\mathit{eff}_{\mathcal{E}}(\textsc{Trm}, \overline{e}) = \bigsqcup e_i$. The proof is straightforward by commutativity of $\sqcup$.

□

3 Canonical forms and value typing environment

We now introduce two additional lemmas that are used in the preservation and soundness proofs.

Lemma 5. *Canonical forms.*
1. *If $\Gamma; f \vdash v : T \mathbin{!} \bot$ and $T <: T_1 \overset{e}{\Rightarrow} T_2$, then $v = (x : T_1') \Rightarrow t$.*
2. *If $\Gamma; f \vdash v : T \mathbin{!} \bot$ and $T <: T_1 \overset{e}{\rightarrow} T_2$, then $v = (x : T_1') \rightarrow t$.*

Proof (Lemma 5). There are only two kinds of values in the language: monomorphic and effect-polymorphic function abstractions. In the first case, v has a monomorphic function type. By inspecting the subtyping rules, it cannot be a subtype of a polymorphic function type, which validates the first case. Similar for the second case. □

Lemma 6. *Environment for type-checking values.*
1. *If $\Gamma; f \vdash v : T \mathbin{!} \bot$, then $\Gamma; f' \vdash v : T \mathbin{!} \bot$ for an arbitrary f'.*
2. *If $\Gamma; f \vdash x : T \mathbin{!} \bot$ for a parameter $x \in \Gamma$, then $\Gamma; f' \vdash x : T \mathbin{!} \bot$ for an arbitrary f'.*

3. *If* $\Gamma; f \vdash \texttt{throw}(p) : \text{Nothing} \; ! \; e$*, then* $\Gamma; f' \vdash \texttt{throw}(p) : \text{Nothing} \; ! \; e$ *for an arbitrary* f'*.*

This lemma states that the polymorphism environment f does not have an impact on type-checking a value, a parameter or an error term.

Proof (Lemma 6). There are two kinds of values: monomorphic and polymorphic function abstractions. Inspecting the corresponding typing rules T-Abs-Mono and T-Abs-Poly, one can see that both rules do not make use of the effect-polymorphism environment f in any way.

For parameters and error terms, the conclusion also follows immediately from the corresponding typing rules. □

4 Substitution Lemmas

We now proof the substitution Lemmas 3 and 4.

As a first step however, we need to introduce a new typing rule for our type system. The reason, as explained in TAPL [1] chapter 16.4, is that the type system has a bottom type (*Nothing*), but no subsumption rule. The typing rules presented in the main paper are therefore incomplete, they do not cover the case of an application expression when the function is an error.

$$\frac{\begin{array}{c}\Gamma; f \vdash t_1 : \mathit{Nothing} \; ! \; e_1 \\ \Gamma; f \vdash t_2 : T_2 \; ! \; e_2\end{array}}{\Gamma; f \vdash t_1 \; t_2 : \mathit{Nothing} \; ! \; \mathit{eff}(\textsc{App}, e_1, e_2, \bot)} \quad \text{(T-App-E)}$$

4.1 Lemma 3

Proof (Lemma 3). The preconditions of the lemma are:

- $\Gamma, x : T_1; f \vdash t : T \; ! \; e_l$ with $f \neq x$
- $\Gamma; g \vdash v : T_2 \; ! \; \bot$
- $T_2 <: T_1$

Proof of $\Gamma; f \vdash t[v \backslash x] : T' \; ! \; e'_l$ with $T' <: T$ and $e'_l \sqsubseteq e_l$ by induction on the typing rules for term t.

♫ Case T-Param: we have $t = z$ and $z : T \in \Gamma, x : T_1$. There are two sub-cases:
  - $z = x$, then $z[v \backslash x] = v$ and $T_1 = T$. Since we have $\Gamma; g \vdash v : T_2 \; ! \; \bot$, by Lemma 6, we have $\Gamma; f \vdash v : T_2 \; ! \; \bot$. The required results are immediate.
  - $z \neq x$, then $z[v \backslash x] = z$. We have $\Gamma, x : T_1; f \vdash z : T \; ! \; e_l$. Since we know that x is not a free variable, $x \notin FV(z)$, we can remove its binding from the typing environment and obtain the result.

- ♫ Case T-Abs-Mono: $t = (y : T_a) \Rightarrow t_1$. From the typing rule we get $\Gamma, x : T_1, y : T_a; \epsilon \vdash t_1 : T_b \ ! \ e$ and $T = T_a \overset{e}{\Rightarrow} T_b$. We can assume that $x \neq y$ and $y \notin FV(v)$.
  By permutation on the environment: $\Gamma, y : T_a, x : T_1; \epsilon \vdash t_1 : T_b \ ! \ e$
  By weakening: $\Gamma, y : T_a; g \vdash v : T_2 \ ! \ \bot$
  Induction hypothesis: $\Gamma, y : T_a; \epsilon \vdash t_1[v\backslash x] : T_b' \ ! \ e'$ with $T_b' <: T_b$ and $e' \sqsubseteq e$
  We have $t[v\backslash x] = (y : T_a) \Rightarrow t_1[v\backslash x]$. By applying T-Abs-Mono we obtain $\Gamma; f \vdash t[v\backslash x] : T_a \overset{e'}{\Rightarrow} T_b' \ ! \ \bot$ and verify $T_a \overset{e'}{\Rightarrow} T_b' <: T$ and $\bot \sqsubseteq e_l$.
- ♫ Case T-App-Mono: $t = t_1 \ t_2$. From the typing rule we have:
  $\Gamma, x : T_1; f \vdash t_1 : T_a \overset{e}{\Rightarrow} T \ ! \ e_1$
  $\Gamma, x : T_1; f \vdash t_2 : T_b \ ! \ e_2$
  $T_b <: T_a$ and $e_l = \mathit{eff}(\text{App}, e_1, e_2, e)$.
  We apply the induction hypothesis to the subterms t_1 and t_2 to obtain
  $\Gamma; f \vdash t_1[v\backslash x] : T_c \ ! \ e_1'$ with $T_c <: T_a \overset{e}{\Rightarrow} T$ and $e_1' \sqsubseteq e_1$
  $\Gamma; f \vdash t_2[v\backslash x] : T_d \ ! \ e_2'$ with $T_d <: T_b$ and $e_2' \sqsubseteq e_2$
  Looking at the subtyping rules, the type T_c can have two possible forms:
  - case $T_c = \mathit{Nothing}$: Applying T-App-E yields $\Gamma; f \vdash t[v\backslash x] : \mathit{Nothing} \, ! \, e_l'$ where $e_l' = \mathit{eff}(\text{App}, e_1', e_2', \bot)$. We have $\mathit{Nothing} <: T$ by S-Nothing.
  - case $T_c = T_a' \overset{e'}{\Rightarrow} T'$: Using S-Trans we obtain $T_d <: T_a'$. Applying T-App-Mono yields $\Gamma; f \vdash t[v\backslash x] : T' \, ! \, e_l'$ where $e_l' = \mathit{eff}(\text{App}, e_1', e_2', e')$. The result $T' <: T$ is immediate.

  In both cases we have $\mathit{eff}(\text{App}, e_1', e_2', e') \sqsubseteq \mathit{eff}(\text{App}, e_1, e_2, e)$ for the resulting effect by monotonicity of *eff*.
- ♫ Case T-Abs-Poly: similar to the case T-Abs-Mono.
- ♫ Case T-App-Param: $t = f \ t_2$. From the typing rule:
  $f : T_a \overset{e}{\Rightarrow} T \in \Gamma, x : T_1$, and $f : T_a \overset{e}{\Rightarrow} T \in \Gamma$ since $f \neq x$
  $\Gamma, x : T_1; f \vdash t_2 : T_b \ ! \ e_2$, and $T_b <: T_a$
  $e_l = \mathit{eff}(\text{App}, \bot, e_2, \bot)$
  By induction hypothesis $\Gamma; f \vdash t_2[v\backslash x] : T_b' \ ! \ e_2'$ with $T_b' <: T_b$, $e_2' \sqsubseteq e_2$.
  Since $f \neq x$, we have $f[v\backslash x] = f$. Therefore, by applying T-App-Param, we obtian:
  $\Gamma; f \vdash t[v\backslash x] : T \ ! \ e_l'$ with $e_l' = \mathit{eff}(\text{App}, \bot, e_2', \bot)$. By monotonicity of *eff*, we obtain $e_l' \sqsubseteq e_l$.
- ♫ Case T-App-Poly: similar to the case T-App-Mono. We need one additional small lemma: if $T' <: T$ then $\mathit{latent}(T') \sqsubseteq \mathit{latent}(T)$. The proof is straightforward.
- ♫ Case T-App-E: similar to T-App-Mono. We make use of the observation that if $T <: \mathit{Nothing}$, then $T = \mathit{Nothing}$ (by analysis of the subtyping rules).
- ♫ Case T-Throw: straightforward.
- ♫ Case T-Try: $t = \texttt{try} \ t_1 \ \texttt{catch}(\overline{p}) \ t_2$. By the typing rule we have:
  $\Gamma, x : T_1; f \vdash t_1 : T_a \ ! \ e_1$ and $T_a <: T$
  $\Gamma, x : T_1; f \vdash t_2 : T_b \ ! \ e_1$ and $T_b <: T$
  $e_l = \mathit{eff}(\text{Catch}(\overline{p}), \mathit{eff}(\text{Try}, e_1), e_2)$
  By applying the induction hypothesis to t_1 and t_2:
  $\Gamma; f \vdash t_1[v\backslash x] : T_a' \ ! \ e_1'$ with $T_a' <: T_a$ and $e_1' \sqsubseteq e_1$

$\Gamma; f \vdash t_2[v\backslash x] : T_b' \mathrel{!} e_2'$ with $T_b' <: T_b$ and $e_2' \sqsubseteq e_2$
Applying T-Try, we obtain
$\Gamma; f \vdash t[v\backslash x] : T \mathrel{!} e_l'$ with $e_l' = \mathit{eff}(\textsc{Catch}(\overline{p}), \mathit{eff}(\textsc{Try}, e_1'), e_2')$.
By monotonicity of *eff* we obtain $e_l' \sqsubseteq e_l$.

□

4.2 Lemma 4

Proof (Lemma 4). The preconditions of the lemma are:

- $\Gamma, x : T_1; x \vdash t : T \mathrel{!} e_l$
- $\Gamma; g \vdash v : T_2 \mathrel{!} \bot$
- $T_2 <: T_1$

Proof of $\Gamma; \epsilon \vdash t[v\backslash x] : T' \mathrel{!} e_l'$ with $T' <: T$ and $e_l' \sqsubseteq e_l \sqcup \mathit{latent}(T_2)$ by induction on the typing rules for term t.

♪ Case T-Param: Similar to the case in Lemma 3.
♪ Case T-Abs-Mono: $t = (y : T_a) \Rightarrow t_1$. From the typing rule we get:
$\Gamma, x : T_1, y : T_a; \epsilon \vdash t_1 : T_b \mathrel{!} e$
$T = T_a \overset{e}{\Rightarrow} T_b$.
We can assume that $x \neq y$ and $y \notin FV(v)$.
By permutation on the environment: $\Gamma, y : T_a, x : T_1; \epsilon \vdash t_1 : T_b \mathrel{!} e$
By weakening: $\Gamma, y : T_a; g \vdash v : T_2 \mathrel{!} \bot$
Applying Lemma 3: $\Gamma, y : T_a; \epsilon \vdash t_1[v\backslash x] : T_b' \mathrel{!} e'$ with $T_b' <: T_b$ and $e' \sqsubseteq e$.
By applying T-Abs-Mono: $\Gamma; \epsilon \vdash t[v\backslash x] : T_a \overset{e'}{\Rightarrow} T_b' \mathrel{!} \bot$.
Verifying $T_a \overset{e'}{\Rightarrow} T_b' <: T$ and $\bot \sqsubseteq e_l$ is straightforward.
♪ Case T-App-Mono: $t = t_1\ t_2$. From the typing rule we have:
$\Gamma, x : T_1; x \vdash t_1 : T_a \overset{e}{\Rightarrow} T \mathrel{!} e_1$
$\Gamma, x : T_1; x \vdash t_2 : T_b \mathrel{!} e_2$
$T_b <: T_a$ and $e_l = \mathit{eff}(\textsc{App}, e_1, e_2, e)$.
We apply the induction hypothesis to the subterms t_1 and t_2 to obtain
$\Gamma; \epsilon \vdash t_1[v\backslash x] : T_c \mathrel{!} e_1'$ with $T_c <: T_a \overset{e}{\Rightarrow} T$ and $e_1' \sqsubseteq e_1 \sqcup \mathit{latent}(T_2)$
$\Gamma; \epsilon \vdash t_2[v\backslash x] : T_d \mathrel{!} e_2'$ with $T_d <: T_b$ and $e_2' \sqsubseteq e_2 \sqcup \mathit{latent}(T_2)$
Looking at the subtyping rules, the type T_c can have two possible forms:
- case $T_c = \mathit{Nothing}$: Applying T-App-E yields $\Gamma; \epsilon \vdash t[v\backslash x] : \mathit{Nothing} \mathrel{!} e_l'$ where $e_l' = \mathit{eff}(\textsc{App}, e_1', e_2', \bot)$. We have $\mathit{Nothing} <: T$ by S-Nothing.
- case $T_c = T_a' \overset{e'}{\Rightarrow} T'$: Using S-Trans we obtain $T_d <: T_a'$. Applying T-App-Mono yields $\Gamma; \epsilon \vdash t[v\backslash x] : T' \mathrel{!} e_l'$ where $e_l' = \mathit{eff}(\textsc{App}, e_1', e_2', e')$. The result $T' <: T$ is immediate.

We need to verify $\mathit{eff}(\textsc{App}, e_1', e_2', e') \sqsubseteq \mathit{eff}(\textsc{App}, e_1, e_2, e) \sqcup \mathit{latent}(T_2)$. Monotonicity of *eff* gives
$\mathit{eff}(\textsc{App}, e_1', e_2', e') \sqsubseteq \mathit{eff}(\textsc{App}, e_1 \sqcup \mathit{latent}(T_2), e_2 \sqcup \mathit{latent}(T_2), e)$
Using the second property of the monotonicity Lemma 1 we conclude:
$\ldots \sqsubseteq \mathit{eff}(\textsc{App}, e_1, e_2, e) \sqcup \mathit{latent}(T_2)$
♪ Case T-Abs-Poly: similar to the case T-Abs-Mono.

♪ Case T-App-Param: $t = x\ t_2$. From the typing rule:
$x : T_a \overset{e}{\Rightarrow} T \in \Gamma, x : T_1$, therefore $T_1 = T_a \overset{e}{\Rightarrow} T$
$\Gamma, x : T_1; x \vdash t_2 : T_b \ !\ e_2$, and $T_b <: T_a$
$e_l = \mathit{eff}(\text{App}, \bot, e_2, \bot)$
By the induction hypothesis $\Gamma; \epsilon \vdash t_2[v\backslash x] : T_b' \ !\ e_2'$ with $T_b' <: T_b$ and $e_2' \sqsubseteq e_2 \sqcup \mathit{latent}(T_2)$.
We have $t[v\backslash x] = v\ t_2[v\backslash x]$. By Lemma 6, we have $\Gamma; \epsilon \vdash v : T_2 \ !\ \bot$. Since $T_2 <: T_1 = T_a \overset{e}{\Rightarrow} T$, there are two possible shapes for T_2:

- case $T_2 = \mathit{Nothing}$: Applying T-App-E yields $\Gamma; \epsilon \vdash t[v\backslash x] : \mathit{Nothing} \ !\ e_l'$ where $e_l' = \mathit{eff}(\text{App}, \bot, e_2', \bot)$. We have $\mathit{Nothing} <: T$ by S-Nothing.
- case $T_2 = T_a' \overset{e'}{\Rightarrow} T'$: Using S-Trans we obtain $T_b' <: T_a'$. Applying T-App-Mono yields $\Gamma; \epsilon \vdash t[v\backslash x] : T' \ !\ e_l'$ where $e_l' = \mathit{eff}(\text{App}, \bot, e_2', e')$. The result $T' <: T$ is immediate.

We need to verify $\mathit{eff}(\text{App}, \bot, e_2', e') \sqsubseteq \mathit{eff}(\text{App}, \bot, e_2, \bot) \sqcup \mathit{latent}(T_2)$. Monotonicity of eff gives
$\mathit{eff}(\text{App}, \bot, e_2', e') \sqsubseteq \mathit{eff}(\text{App}, \bot, e_2 \sqcup \mathit{latent}(T_2), e')$
Since $e' \sqsubseteq \bot \sqcup e'$, monotonicity gives
$\ldots \sqsubseteq \mathit{eff}(\text{App}, \bot, e_2 \sqcup \mathit{latent}(T_2), \bot \sqcup e')$
Using the second property of the monotonicity Lemma 1
$\ldots \sqsubseteq \mathit{eff}(\text{App}, \bot, e_2, \bot) \sqcup e' \sqcup \mathit{latent}(T_2)$
And finally, since $e' = \mathit{latent}(T_2)$, we obtain
$\ldots \sqsubseteq \mathit{eff}(\text{App}, \bot, e_2, \bot) \sqcup \mathit{latent}(T_2)$.

♪ Case T-App-Poly: similar to the case T-App-Mono. We again use the property that if $T' <: T$, then $\mathit{latent}(T') \sqsubseteq \mathit{latent}(T)$.

♪ Case T-App-E: similar to T-App-Mono. We need to make use of the observation that if $T <: \mathit{Nothing}$, then $T = \mathit{Nothing}$ (by analysis of the subtyping rules).

♪ Case T-Throw: straightforward.

♪ Case T-Try: $t = \texttt{try}\ t_1\ \texttt{catch}(\overline{p})\ t_2$. By the typing rules:
$\Gamma, x : T_1; x \vdash t_1 : T_a \ !\ e_1$ and $T_a <: T$
$\Gamma, x : T_1; x \vdash t_2 : T_b \ !\ e_1$ and $T_b <: T$
$e_l = \mathit{eff}(\text{Catch}(\overline{p}), \mathit{eff}(\text{Try}, e_1), e_2)$
By applying the induction hypothesis to t_1 and t_2:
$\Gamma; \epsilon \vdash t_1[v\backslash x] : T_a' \ !\ e_1'$ with $T_a' <: T_a$ and $e_1' \sqsubseteq e_1 \sqcup \mathit{latent}(T_2)$
$\Gamma; \epsilon \vdash t_2[v\backslash x] : T_b' \ !\ e_2'$ with $T_b' <: T_b$ and $e_2' \sqsubseteq e_2 \sqcup \mathit{latent}(T_2)$
Applying T-Try, we obtain $\Gamma; \epsilon \vdash t[v\backslash x] : T \ !\ e_l'$ with $e_l' = \mathit{eff}(\text{Catch}(\overline{p}), \mathit{eff}(\text{Try}, e_1'), e_2')$.
By the monotonicity Lemma 1 we obtain $e_l' \sqsubseteq e_l$.

$\square$

5 Soundness theorems

5.1 Preservation (Theorem 1)

Proof (Theorem 1). The preconditions are

- $\Gamma, f \vdash t : T \ ! \ e$
- $t \Downarrow \langle r, S \rangle$

Proof of $\Gamma, f \vdash r : T' \ ! \ e'$ with $T' <: T$ by induction on the evaluation rules for term t.

♩ Case E-App-E1: $t = t_1 \ t_2$. We have
 $t_1 \Downarrow \langle \mathtt{throw}(p), S_1 \rangle \qquad t \Downarrow \langle \mathtt{throw}(p), S \rangle$, where $S = \mathit{dynEff}(S_1, \emptyset, \emptyset)$
 By typing rule T-Throw, we obtain $\Gamma; f \vdash r : \mathit{Nothing} \ ! \ \mathit{eff}(\text{Throw}(p))$.
 The result $\mathit{Nothing} <: T$ is immediate by S-Nothing.

♩ Case E-App-E2: similar.

♩ Case E-App: $t = t_1 \ t_2$. We have:
 $t_1 \Downarrow \langle (x : T_1) \mapsto t_r, S_1 \rangle$
 $t_2 \Downarrow \langle v_2, S_2 \rangle$
 $t_r[v_2 \backslash x] \Downarrow \langle r, S_l \rangle$
 We distinguish sub-cases for the typing rule of the application expression $t_1 \ t_2$:
 - case T-App-E: $\Gamma; f \vdash t_1 : \mathit{Nothing} \ ! \ e_1$. By induction hypothesis, we have $\Gamma; f \vdash (x : T_1) \mapsto t_r : T_1' \ ! \ e_1'$ such that $T_1' <: \mathit{Nothing}$. Since the type of a function abstraction cannot be *Nothing*, this case is impossible.
 - case T-App-Param: $t_1 = p$ for some parameter p. This is impossible, since we have $t_1 \Downarrow \langle (x : T_1) \mapsto t_r, S_1 \rangle$, but there is no evaluation rule for parameters.
 - case T-App-Mono: we have
 $\Gamma; f \vdash t_1 : T_a \stackrel{e_l}{\Rightarrow} T \ ! \ e_1$
 $\Gamma; f \vdash t_2 : T_b \ ! \ e_2$ with $T_b <: T_a$
 Applying the induction hypothesis to t_1 and t_2:
 $\Gamma; f \vdash (x : T_1) \mapsto t_r \ ! \ T_c \ ! \ e_1'$ with $T_c <: T_a \stackrel{e_l}{\Rightarrow} T$
 $\Gamma; f \vdash v_2 : T_b' \ ! \ \bot$ with $T_b' <: T_b$
 According to the subtyping rules, T_c can either be *Nothing* or a monomorphic function type. But since T_c is the type of a function abstraction, it cannot be *Nothing*, therefore we have $T_c = T_d \stackrel{e_l'}{\Rightarrow} T_e$. By the canonical forms Lemma 5 the corresponding function abstraction is a monomorphic one. We obtain:
 $\Gamma; f \vdash (x : T_1) \Rightarrow t_r \ ! \ T_1 \stackrel{e_l'}{\Rightarrow} T_e \ ! \ e_1'$, with $T_d = T_1$
 Therefore:
 $\Gamma, x : T_1; \epsilon \vdash t_r : T_e \ ! \ e_l'$
 By transitivity of subtyping, we have $T_b' <: T_b <: T_a <: T_1$ and we can apply substitution Lemma 3 to obtain
 $\Gamma; \epsilon \vdash t_r[v_2 \backslash x] : T_e' \ ! \ e_l''$ with $T_e' <: T_e$.

Now we apply the induction hypothesis on $t_r[v_2\backslash x]$ to obtain
$\Gamma; \epsilon \vdash r : T_r \ ! \ e_r$ with $T_r <: T'_e$.
By Lemma 6 we get $\Gamma; f \vdash r : T_r \ ! \ e_r$, and by transitivity of subtyping $T_r <: T'_e <: T_e <: T$.

  - case T-App-Mono: similar.

- Case E-Throw: $t = \texttt{throw}(p)$. By T-Throw, we have
  $\Gamma; f \vdash \texttt{throw}(p) : \mathit{Nothing} \ ! \ \mathit{eff}(\textsc{Throw}(p))$.
  The same typing rule is also applied to the result r, and we verify $\mathit{Nothing} <: \mathit{Nothing}$ by S-Refl.
- Case E-Try-E: $t = \texttt{try} \ t_1 \ \texttt{catch}(\overline{p}) \ t_2$. We have
  $t_1 \Downarrow \langle \mathit{throw}(p), S_1 \rangle$
  $t_2 \Downarrow \langle r_2, S_2 \rangle$
  From the typing rule T-Try:
  $\Gamma; f \vdash t_1 : T1 \ ! \ e_1$ with $T_1 <: T$
  $\Gamma; f \vdash t_2 : T2 \ ! \ e_2$ with $T_2 <: T$
  Applying the induction hypothesis to t_1, we obtain $\Gamma; f \vdash r_2 : T'_2 \ ! \ e'_2$ with $T'_2 <: T_2$. Since $r = r_2$, in remains to verify using S-Trans that $T'_2 <: T_2 <: T$.
- Case E-Try: similar.

□

5.2 Effect soundness (Theorem 2)

Proof (Theorem 2). The preconditions are

- $\Gamma, f \vdash t : T \ ! \ e$
- $t \Downarrow \langle r, S \rangle$

Proof of $S \preceq e \sqcup \mathit{latent}(\Gamma(f))$ by induction on the evaluation rules for term t.

- Case T-App-E1: $t = t_1 \ t_2$. We have
  $t_1 \Downarrow \langle \texttt{throw}(p), S_1 \rangle$
  $t \Downarrow \langle \texttt{throw}(p), \mathit{dynEff}(\textsc{App}, S_1, \emptyset, \emptyset) \rangle$
  We look at the sub-cases corresponding to the typing rules for applications:
  - case T-App-E: $\Gamma; f \vdash t_1 : \mathit{Nothing} \ ! \ e_1$ and $\Gamma; f \vdash t_2 : T_2 \ ! \ e_2$ and $e = \mathit{eff}(\textsc{App}, e_1, e_2, \bot)$.
    By induction hypothesis, we have $S_1 \preceq e_1 \sqcup \mathit{latent}(\Gamma(f))$. Note that trivially, $\emptyset \preceq e_x$ for any e_x. Therefore we can apply the consistency Lemma 2 to obtain $\mathit{dynEff}(\textsc{App}, S_1, \emptyset, \emptyset) \preceq \mathit{eff}(\textsc{App}, e_1, e_2, \bot) \sqcup \mathit{latent}(\Gamma(f))$.
  - case T-App-Param: $t_1 = p$ for some parameter p. This case is not possible because there is no evaluation rule for parameters.
  - case T-App-Mono:
    $\Gamma; f \vdash t_1 : T_1 \overset{e_l}{\Rightarrow} T \ ! \ e_1$
    $\Gamma; f \vdash t_2 : T_2 \ ! \ e_2$ with $T_2 <: T_1$
    $e = \mathit{eff}(\mathit{App}, e_1, e_2, e_l)$
    The induction hypothesis on t_1 gives
    $S_1 \preceq e_1 \sqcup \mathit{latent}(\Gamma(f))$

Since $\emptyset \preceq e_x$ for any e_x, we can apply Lemma 2 to obtain
$dynEff(\textsc{App}, S_1, \emptyset, \emptyset) \preceq eff(\textsc{App}, e_1, e_2, e_l) \sqcup latent(\Gamma(f))$.

- case T-App-Poly: similar.

♩ Case E-App-E2: similar

♩ Case E-App: Again, $t = t_1\ t_2$. We have the following preconditions
$t_1 \Downarrow \langle (x : T_a) \mapsto t_r, S_1\rangle$
$t_2 \Downarrow \langle v_2, S_2\rangle$
$t_r[v_2\backslash x] \Downarrow \langle r, S_l\rangle$
$t \Downarrow \langle r, S\rangle$ with $S = dynEff(\textsc{App}, S_1, S_2, S_l)$
There is a sub-case for every possible typing rule for the application expression.

- case T-App-E: $\Gamma; f \vdash t_1 : Nothing\ !\ e_1$. By the preservation theorem, we have $\Gamma; f \vdash (x : T_a) \mapsto t_r : T_1\ !\ e_1'$ with $T_1' <: Nothing$, which cannot be derived with any typing rule. Therefore, this case is impossible.
- case T-App-Param: $t_1 = p$ for some parameter p. This case is not possible because there is no evaluation rule for parameters.
- case T-App-Mono:
  $\Gamma; f \vdash t_1 : T_1 \overset{e_l}{\Rightarrow} T\ !\ e_1$
  $\Gamma; f \vdash t_2 : T_2\ !\ e_2$ with $T_2 <: T_1$
  $e = eff(\textsc{App}, e_1, e_2, e_l)$
  By induction hypothesis on t_1 and t_2, we obtain
  $S_1 \preceq e_1 \sqcup latent(\Gamma(f))$
  $S_2 \preceq e_2 \sqcup latent(\Gamma(f))$
  By the preservation theorem, we have $\Gamma; f \vdash (x : T_a) \mapsto t_r : T_1'\ !\ \bot$ with $T_1' <: T_1 \overset{e_l}{\Rightarrow} T$. Since the term is a function abstraction, $T_1' = Nothing$ is not possible, which implies $T_1' = T_a \overset{e_l'}{\Rightarrow} T'$. By the canonical forms Lemma 5, the term is a monomorphic function abstraction:
  $\Gamma; f \vdash (x : T_a) \Rightarrow t_r : T_a \overset{e_l'}{\Rightarrow} T'\ !\ \bot$
  Therefore, we have $\Gamma, x : T_a; \epsilon \vdash t_r : T'\ !\ e_l'$. Applying preservation on t_2 gives us $\Gamma; f \vdash v_2 : T_2'\ !\ \bot$ with $T_2' <: T_2 <: T_1 <: T_a$. We can apply substitution Lemma 3 to obtain
  $\Gamma; \epsilon \vdash t_r[v_2\backslash x] : T''\ !\ e_l''$ with $T'' <: T'$ and $e_l'' \sqsubseteq e_l'$.
  The induction hypothesis on $t[v_2\backslash x]$ gives $S_l \preceq e_l'' \sqcup latent(\Gamma(\epsilon))$. Since $latent(\Gamma(\epsilon)) = \bot$, and by $e_l'' \sqsubseteq e_l' \sqsubseteq e_l$, we have $S_l \preceq e_l$. Together with the induction hypotheses, we apply the consistency Lemma 2 to obtain
  $S \preceq eff(\textsc{App}, e_1, e_2, e_l) \sqcup latent(\Gamma(f))$
- case T-App-Poly: similar (see main paper).

♩ Case E-Throw: $t = \texttt{throw}(p)$ and $t \Downarrow \langle \texttt{throw}(p), dynEff(\textsc{Throw}(p))\rangle$.
By typing rule T-Throw, we obtain
$\Gamma; f \vdash \texttt{throw}(p) : Nothing\ !\ eff(\textsc{Throw}(p))$.
By Lemma 2, we conclude
$dynEff(\textsc{Throw}(p)) \preceq eff(\textsc{Throw}(p)) \sqcup latent(\Gamma(f))$.

♩ Case E-Try-E: $t = \texttt{try}\ t_1\ \texttt{catch}(\overline{p})\ t_2$. We have
$t_1 \Downarrow \langle throw(p), S_1\rangle$
$t_2 \Downarrow \langle r_2, S_2\rangle$

$S_t = \mathit{dynEff}(\textsc{Try}, S_1) \qquad S = \mathit{dynEff}(\textsc{Catch}(\overline{p}), S_t, S_2)$

From the typing rule T-Try:

$\Gamma; f \vdash t_1 : T1 \mathbin{!} e_1$ with $T_1 <: T$

$\Gamma; f \vdash t_2 : T2 \mathbin{!} e_2$ with $T_2 <: T$

$e_t = \mathit{eff}(\textsc{Try}, e_1) \qquad e = \mathit{eff}(\textsc{Catch}(\overline{p}), e_t, e_2)$

The induction hypotheses are

$S_1 \preceq e_1 \sqcup \mathit{latent}(\Gamma(f))$

$S_2 \preceq e_2 \sqcup \mathit{latent}(\Gamma(f))$

Applying Lemma 2 to S_t and e_t yields $S_t \preceq e_t \sqcup \mathit{latent}(\Gamma(f))$. Now we can apply the same lemma to S and e and conclude $S \preceq e \sqcup \mathit{latent}(\Gamma(f))$.

- Case E-Try: similar.

$\square$

References

1. Benjamin C. Pierce. *Types and programming languages.* MIT Press, 2002.
2. Lukas Rytz and Martin Odersky. Lightweight polymorphic effects. Technical report, EPFL, 2012.