

1 Checked Error Handling

In this exercise we use the Simply-Typed Lambda Calculus (STLC) extended with rules for error handling. In this language, terms may reduce to a normal form **error**, which is *not* a value. In addition, we add the new term form **try** t_1 **with** t_2 , which allows handling errors that occur while evaluating t_1 .

Here is a summary of the extensions to syntax and evaluation:

$t ::=$	terms :
\dots	
error	run-time error
try t with t	trap errors

New evaluation rules:

$$\begin{array}{ll}
 \text{(E-APPERR1)} \quad \mathbf{error} \ t_2 \longrightarrow \mathbf{error} & \text{(E-APPERR2)} \quad v_1 \ \mathbf{error} \longrightarrow \mathbf{error} \\
 \text{(E-TRYVALUE)} \quad \mathbf{try} \ v_1 \ \mathbf{with} \ t_2 \longrightarrow v_1 & \text{(E-TRYERROR)} \quad \mathbf{try} \ \mathbf{error} \ \mathbf{with} \ t_2 \longrightarrow t_2 \\
 \text{(E-TRY)} \quad \frac{t_1 \longrightarrow t'_1}{\mathbf{try} \ t_1 \ \mathbf{with} \ t_2 \longrightarrow \mathbf{try} \ t'_1 \ \mathbf{with} \ t_2}
 \end{array}$$

(Note that these extensions are exactly those summarized in Figures 14-1 and 14-2 on pages 172 and 174 of the TAPL book. However, also note that we will use *different* typing rules.)

The goal of this exercise is to prove that the following progress theorem holds:

If $\emptyset ; \mathbf{false} \vdash t : T$, then either t is a value or else $t \longrightarrow t'$.

The above theorem uses a typing judgment extended with a Boolean value E , written $\Gamma ; E \vdash t : T$ where $E \in \{\mathbf{true}, \mathbf{false}\}$. The theorem says that a well-typed term that is closed (that is, it does not have free variables, which is expressed using $\Gamma = \emptyset$) is either a value, or else it can be reduced *as long as* $E = \mathbf{false}$.

Intuitively, the predicate E in the typing judgment $\Gamma ; E \vdash t : T$ determines whether the term t is in an *impure* position, i.e. whether it is allowed to reduce to **error** or not. The typing rules for the error (handling) terms are as follows:

$$\begin{array}{ll}
 \text{(T-ERROR)} \quad \Gamma ; \mathbf{true} \vdash \mathbf{error} : T & \text{(T-TRY)} \quad \frac{\Gamma ; \mathbf{true} \vdash t_1 : T \quad \Gamma ; E \vdash t_2 : T}{\Gamma ; E \vdash \mathbf{try} \ t_1 \ \mathbf{with} \ t_2 : T}
 \end{array}$$

The remaining typing rules (those inherited from STLC), simply “push” the impurity predicate E unchanged from the conclusion into the premises. Intuitively, if a term is allowed to produce an error, all its sub-terms are also allowed to do so.

$$\begin{array}{ll}
 \text{(T-VAR)} \quad \frac{x : T \in \Gamma}{\Gamma ; E \vdash x : T} & \text{(T-ABS)} \quad \frac{\Gamma \ x : T_1 ; E \vdash t_2 : T_2}{\Gamma ; E \vdash \lambda x : T_1. t_2 : T_1 \rightarrow T_2} \\
 \text{(T-APP)} \quad \frac{\Gamma ; E \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma ; E \vdash t_2 : T_1}{\Gamma ; E \vdash t_1 \ t_2 : T_2}
 \end{array}$$

Your task is as follows:

1. Prove the above progress theorem using structural induction. (You can use the canonical forms lemma for STLC as seen in the lecture without proof.)

2. How does E contribute to your proof? Where would your proof “go wrong” if we removed it from the typing rules?
3. (*Extra effort*) Try to prove the preservation theorem as well. Is it possible with the above typing rules? If not, is there a way to alter the typing rules to make the theorem hold? In either case, it may be useful to slightly change the form of the function type.

1.1 Solution

Progress of pure terms. We need to prove a more general statement first, namely a modified version of progress that states that closed, well-typed terms in normal form are either values or errors.

Lemma 1. *If $\emptyset ; E \vdash t : T$, then either (a) t is a value, (b) $t = \mathbf{error}$, or (c) $t \longrightarrow t'$.*

Note that this lemma says nothing about the truth-value of the predicate E , it holds for both $E = \mathbf{true}$ and $E = \mathbf{false}$. In other words, the lemma should still hold for a simpler typing judgment $\Gamma \vdash t : T$, where we simply ignore E . This is exactly the typing judgment shown in Fig. 14-2 on p. 174 of TAPL, and the corresponding progress theorem carries over essentially unchanged.

Proof. The proof is by induction on typing derivations and most cases are essentially identical to those of the proof of Thm. 9.3.5, in TAPL (p. 105). We only show the cases for (T-APP) and (T-TRY) here.

For (T-APP), we have $t = t_1 t_2$, $\emptyset ; E \vdash t_1 : T_1 \rightarrow T_2$ and $\emptyset ; E \vdash t_2 : T_1$. By the IH, either t_1 is (a) a value, (b) an error, or (c) takes a step, and likewise for t_2 . If t_1 can take a step, then (E-APP1) applies to t ; if t_1 is an error then (E-APPERR1) applies instead. If t_1 is a value, but t_2 can take a step or is an error, then either (E-APP2) or (E-APPERR2) applies, respectively. Finally, if both t_1 and t_2 are values, then the canonical forms lemma tells us that $t_1 = \lambda x : T_1.t_3$ for some t_3 , so that (E-APPABS) applies to t .

For (T-TRY), we have $t = \mathbf{try } t_1 \mathbf{ with } t_2$, $\emptyset ; \mathbf{true} \vdash t_1 : T$ and $\emptyset ; E \vdash t_2 : T$, so by the IH, t_1 is either (a) a value, (b) an error, or (c) takes a step. In each case, we can take a step using, respectively, the rules (E-TRYVALUE), (E-TRYERROR) or (E-TRY). \square

With the above lemma, we can now proof the progress theorem for pure terms.

Proof. The proof is by induction on typing derivations and resembles again that of the progress theorem for the STLC (see TAPL, Thm. 9.3.5, p. 105).

The cases (T-VAR), (T-ABS) and (T-APP) are again identical to those in the proof for STLC (modulo the appearance of $E = \mathbf{false}$ in the context) since the corresponding rules push the impurity predicate unchanged into the IH.

The case (T-ERROR) cannot occur since, by definition, its context is impure, i.e. $E = \mathbf{true}$. The only remaining case is (T-TRY). We have $t = \mathbf{try } t_1 \mathbf{ with } t_2$, $\emptyset ; \mathbf{true} \vdash t_1 : T$ and $\emptyset ; \mathbf{false} \vdash t_2 : T$. By the above lemma, t_1 is either (a) a value, (b) an error or (c) takes a step. Again, we can take a step in each case, using, respectively, the rules (E-TRYVALUE), (E-TRYERROR) or (E-TRY). \square

2 The call-by-value simply typed lambda calculus with returns

Consider a variant of the call-by-value simply typed lambda calculus extended to support a new language construct: **return** t , which immediately returns a given term t from an **enclosing** lambda, disregarding any potential further computation typically needed for call-by-value evaluation rules.

The grammar of the extension is defined as follows:

$$\begin{aligned} v &::= \lambda x : T . t \mid \mathbf{true} \mid \mathbf{false} & (\text{values}) \\ t &::= x \mid v \mid t \ t \mid \mathbf{return} \ t \mid \mathbf{frame} \ t & (\text{terms}) \\ T &::= \mathbf{Bool} \mid T \rightarrow T & (\text{types}) \end{aligned}$$

There are two additional forms for terms: the *return* form **return** t and the *frame* form **frame** t . The latter is not meant to be used in terms representing source programs. Rather, it is introduced with the following reduction rule:

$$(\mathbf{E-APP}) \ (\lambda x : T . t) \ v \longrightarrow \mathbf{frame} \ ([x \mapsto v]t)$$

The frame form marks the scope from which local returns should return. If the body of a frame reduces to a value, the frame should be removed. Otherwise, a return form should stop propagating at a frame in much the same way that an error stops at a **try/with**.

In the previous exercise, we extended the typing rules to track impurity. We do something similar here, but to track the types of values that can be *returned* from a term. That is in addition to the normal type of the term, which is the one we get if it finishes evaluating without encountering a **return**. We use \mathbf{R} to denote a *set* of types, i.e., $\{T_1, \dots, T_n\}$.

$$(\mathbf{T-TRUE}) \ \Gamma \vdash \mathbf{true} : \mathbf{Bool} \mid \emptyset \qquad (\mathbf{T-FALSE}) \ \Gamma \vdash \mathbf{false} : \mathbf{Bool} \mid \emptyset$$

$$(\mathbf{T-VAR}) \ \frac{x : T \in \Gamma}{\Gamma \vdash x : T \mid \emptyset}$$

$$(\mathbf{T-ABS}) \ \frac{\Gamma, x : T_1 \vdash t_2 : T_2 \mid R_2 \quad R_2 \subseteq \{T_2\}}{\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \rightarrow T_2 \mid \emptyset}$$

$$(\mathbf{T-APP}) \ \frac{\Gamma \vdash t_1 : T_1 \rightarrow T_2 \mid R_1 \quad \Gamma \vdash t_2 : T_1 \mid R_2}{\Gamma \vdash t_1 \ t_2 : T_2 \mid (R_1 \cup R_2)}$$

$$(\mathbf{T-RETURN}) \ \frac{\Gamma \vdash t_1 : T_1 \mid R_1}{\Gamma \vdash \mathbf{return} \ t_1 : T \mid (R_1 \cup \{T_1\})}$$

$$(\mathbf{T-FRAME}) \ \frac{\Gamma \vdash t_1 : T \mid R_1 \quad R_1 \subseteq \{T\}}{\Gamma \vdash \mathbf{frame} \ t_1 : T \mid \emptyset}$$

Note that the conclusion of **T-RETURN** mentions a T that does not appear anywhere in the premises. What does that mean?

In this exercise, you are to adjust the existing evaluation rules such that they correctly and comprehensively describe the semantics of the extension. In particular, **return** terms must short-circuit most of the terms, much like **error** terms did in the previous exercise.

You must then prove the preservation theorem for the system.

More precisely, you have the following tasks:

1. Extend the evaluation rules (by adding new rules and/or changing existing ones) to express the early return semantics provided by `return`. Identify the evaluation strategy used by the specification and make sure that your extension adheres to it. Make sure that **return** terms propagate like **error** terms did in the previous exercise, so that progress would hold.
2. Prove the following two lemmas:
 Values are pure: If $\Gamma \vdash v : T \mid R$, then $R = \emptyset$.
return is polymorphic: If $\Gamma \vdash \text{return } t_1 : T \mid R$, then $\Gamma \vdash \text{return } t_1 : U \mid R$.
3. State, but do not prove, the weakening lemma, the permutation lemma, and the substitution lemma for this system.
4. State and prove the preservation theorem. You may rely on the above lemmas as well as the usual inversion lemma and canonical forms lemma. Think about how R evolves when we take one step of evaluation: does it always remain the same?
5. (*Extra effort*) State and prove the progress theorem as well. The statement should exclude top-level returns – intuitively, a closed term with a “naked” return doesn’t make sense.

Before you begin, think carefully about the following simple term: $\lambda x : \text{Bool}. (\text{return true}) x$. Intuitively, it makes sense. Once this lambda is applied, it is going to evaluate to **true**, regardless of the input. What type or types will be assigned to **return true**?

2.1 Solution

2.1.1 Evaluation rules

$$\frac{t_1 \longrightarrow t'_1}{t_1 \ t_2 \longrightarrow t'_1 \ t_2} \quad (\text{E-APP1})$$

$$\frac{t_2 \longrightarrow t'_2}{v_1 \ t_2 \longrightarrow v_1 \ t'_2} \quad (\text{E-APP2})$$

$$(\lambda x : T_1 . t_2) \ v_1 \longrightarrow \text{frame } ([x \mapsto v_1]t_2) \quad (\text{E-APP})$$

$$\frac{t_1 \longrightarrow t'_1}{\text{return } t_1 \longrightarrow \text{return } t'_1} \quad (\text{E-RET})$$

$$(\text{return } v_1) \ t_2 \longrightarrow \text{return } v_1 \quad (\text{E-APPRET1})$$

$$v_1 \ (\text{return } v_2) \longrightarrow \text{return } v_2 \quad (\text{E-APPRET2})$$

$$\text{return } (\text{return } v_1) \longrightarrow \text{return } v_1 \quad (\text{E-RETRRET})$$

$$\frac{t_1 \longrightarrow t'_1}{\text{frame } t_1 \longrightarrow \text{frame } t'_1} \quad (\text{E-FRAME})$$

$$\text{frame } v_1 \longrightarrow v_1 \quad (\text{E-FRAMEVAL})$$

$$\text{frame } (\text{return } v_1) \longrightarrow v_1 \quad (\text{E-FRAMERET})$$

2.1.2 Values are pure

Values-are-pure lemma: If $\Gamma \vdash v : T \mid R$, then $R = \emptyset$.

Proof: immediate from T-TRUE, T-FALSE and T-ABS.

2.1.3 return is polymorphic

Return-is-poly lemma: If $\Gamma \vdash \mathbf{return} \ t_1 : T \mid R$, then $\Gamma \vdash \mathbf{return} \ t_1 : U \mid R$.

Proof: from T-RETURN, we know there must exist T_1 and R_1 such that $\Gamma \vdash t_1 : T_1 \mid R_1$ with $R = R_1 \cup \{T_1\}$. Then, by T-RETURN, we have $\Gamma \vdash \mathbf{return} \ t_1 : U \mid R_1 \cup \{T_1\}$.

2.1.4 Additional lemmas

Weakening lemma: If $\Gamma \vdash t : T \mid R$ and $x \notin \text{dom}(\Gamma)$, then $\Gamma, x : T_1 \vdash t : T \mid R$.

Permutation lemma: If $\Gamma \vdash t : T \mid R$ and Δ is a permutation of Γ , then $\Delta \vdash t : T \mid R$.

Substitution lemma: If $\Gamma, x : S \vdash t : T \mid R$ and $\Gamma \vdash s : S \mid \emptyset$, then $\Gamma \vdash [x \mapsto s]t : T \mid R$.

2.1.5 Proof of preservation

Preservation theorem:

If $\Gamma \vdash t : T \mid R$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T \mid R'$ with $R' \subseteq R$.

Proof: by induction on the typing derivation tree. We proceed by case analysis of the last typing rule in the derivation tree.

Case T-TRUE, T-FALSE, T-VAR, T-ABS and T-RETURN Trivial, because they are all for normal forms, i.e., there is no t' such that $t \longrightarrow t'$.

Case T-FRAME $t = \mathbf{frame} \ t_1$, $\Gamma \vdash t_1 : T \mid R_1$, $R_1 \subseteq \{T\}$, $R = \emptyset$.

We proceed by case analysis for the derivation of $t \longrightarrow t'$.

Case E-FRAME: $t_1 \longrightarrow t'_1$, $t' = \mathbf{frame} \ t'_1$. By the induction hypothesis, we have $\Gamma \vdash t'_1 : T \mid R'_1$ with $R'_1 \subseteq R_1$. Then by T-FRAME and because $R'_1 \subseteq R_1 \subseteq \{T\}$, we have $\Gamma \vdash \mathbf{frame} \ t'_1 : T \mid \emptyset$, as required.

Case E-FRAMEVAL: $t_1 = v_1$, $t' = v_1$. By the values-are-pure lemma, we know that $R_1 = \emptyset \subseteq R$. This combines with $\Gamma \vdash t_1 : T \mid R_1$ to conclude.

Case E-FRAMERET: $t_1 = \mathbf{return} \ v_1$, $t' = v_1$. From T-RETURN, we know that there exist T_1 and Q such that $\Gamma \vdash v_1 : T_1 \mid Q$ and $R_1 = Q \cup \{T_1\}$. By the values-are-pure lemma, we know that $Q = \emptyset$, hence $R_1 = \{T_1\}$. Since $R_1 \subseteq \{T\}$, we have $\{T_1\} \subseteq \{T\}$ and hence $T_1 = T$. This allows us to conclude that $\Gamma \vdash v_1 : T \mid \emptyset$, as required.

Case T-RETURN $t = \mathbf{return} \ t_1$, $\Gamma \vdash t_1 : T_1 \mid R_1$, $R = R_1 \cup \{T_1\}$.

We proceed by case analysis for the derivation of $t \longrightarrow t'$.

Case E-RET: $t_1 \longrightarrow t'_1$, $t' = \mathbf{return} \ t'_1$. By the induction hypothesis, we have $\Gamma \vdash t'_1 : T_1 \mid R'_1$ with $R'_1 \subseteq R_1$. Then by T-RETURN, we have $\Gamma \vdash \mathbf{return} \ t'_1 : T_1 \mid (R'_1 \cup \{T_1\})$. Because $R'_1 \subseteq R_1$, we have that $R'_1 \cup \{T_1\} \subseteq R$, which concludes.

Case E-RETRRET: $t_1 = \mathbf{return} \ v_{11}$, $t' = \mathbf{return} \ v_{11}$. By the polymorphic-return lemma, we have $\Gamma \vdash t_1 : T \mid R_1$. We conclude by noting that $t' = t_1$ and $R_1 \subseteq R$.

Case T-APP $t = t_1 \ t_2$, $\Gamma \vdash t_1 : T_1 \rightarrow T \mid R_1$, $\Gamma \vdash t_2 : T_1 \mid R_2$, $R = R_1 \cup R_2$.

We proceed by case analysis for the derivation of $t \longrightarrow t'$.

Case E-APP1: $t_1 \longrightarrow t'_1$, $t' = t'_1 \ t_2$. By the induction hypothesis, we have $\Gamma \vdash t'_1 : T_1 \mid R'_1$ with $R'_1 \subseteq R_1$. Then by T-APP, we have $\Gamma \vdash t'_1 \ t_2 : T \mid (R'_1 \cup R_2)$. Since $R'_1 \subseteq R_1$, we have that $(R'_1 \cup R_2) \subseteq R$, which concludes.

Case E-APP2: Similar.

Case E-APPRET1: $t_1 = \mathbf{return} \ v_1$, $t' = \mathbf{return} \ v_1$. By the polymorphic-return lemma, we have $\Gamma \vdash t_1 : T \mid R_1$. We conclude by noting that $t' = t_1$ and $R_1 \subseteq R$.

Case E-APPRET2: Similar.

Case E-APP: $t_1 = \lambda x : T_{11}. t_{12}$, $t_2 = v_2$, $t' = \mathbf{frame} \ [x \mapsto v_2]t_{12}$. From T-ABS, $T_{11} = T_1$ and $\Gamma, x : T_1 \vdash t_{12} : T \mid Q$ with $Q \subseteq \{T\}$. By the values-are-pure lemma, $R_2 = \emptyset$, i.e., $\Gamma \vdash v_2 : T_1 \mid \emptyset$. Therefore, by the substitution lemma, we have $\Gamma \vdash [x \mapsto v_2]t_{12} : T \mid Q$. Combining with $Q \subseteq \{T\}$, we use T-FRAME to conclude that $\Gamma \vdash \mathbf{frame} \ [x \mapsto v_2]t_{12} : T \mid \emptyset$, as required.

2.1.6 Commentary

The particular choice of typing rules we used is not the only possible solution. The typing rules presented treat R like their output, something that we can determine from the environment, term and the premise.

In (T-VAR), the return set is always empty, because there's nothing else we can reasonably put there. It makes sense when applying the substitution in E-APP, since we replace variables by values, which are pure. In (T-RETURN), we *extend* the current returns with the type of the value we're currently trying to return. In (T-APP), we just take the union of the return sets of the two terms we have. In T-ABS, where we must actually “merge” a set R_2 with a single type T_2 , we use the slight trick of demanding that $R_2 \subseteq \{T_2\}$, which is equivalent to demanding that R_2 be empty or equal to the singleton $\{T_2\}$.

So do we need R to be a set in this task? We want R to sometimes be empty (because some expressions never explicitly **return**), and we want a convenient notion of “extending” R with another type. A set works quite well here. Try to think how you'd reformulate rules such as (T-APP) without using sets.

3 Appendix

3.1 The call-by-value simply typed lambda calculus

The complete reference of the variant of simply typed lambda calculus (with *Bool* ground type representing the type of values *true* and *false*) used in “The call-by-value simply typed lambda calculus with returns” is as follows:

$$\begin{aligned}
v &::= \lambda x : T. t \mid bv && (\text{values}) \\
bv &::= \mathbf{true} \mid \mathbf{false} && (\text{boolean values}) \\
t &::= x \mid v \mid t \ t && (\text{terms}) \\
p &::= t && (\text{programs}) \\
T &::= \mathbf{Bool} \mid T \rightarrow T && (\text{types})
\end{aligned}$$

Evaluation rules:

$$\frac{t_1 \longrightarrow t'_1}{t_1 \ t_2 \longrightarrow t'_1 \ t_2} \quad (\text{E-APP1})$$

$$\frac{t_2 \longrightarrow t'_2}{v_1 \ t_2 \longrightarrow v_1 \ t'_2} \quad (\text{E-APP2})$$

$$(\lambda x : T_1. t_1) \ v_2 \longrightarrow [x \mapsto v_2] t_1 \quad (\text{E-APPABS})$$

Typing rules:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad (\text{T-VAR})$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash (\lambda x : T_1. t_2) : T_1 \rightarrow T_2} \quad (\text{T-ABS})$$

$$\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1 \ t_2 : T_2} \quad (\text{T-APP})$$

$$\frac{}{\Gamma \vdash \mathbf{true} : \mathbf{Bool}} \quad (\text{T-FALSE})$$

$$\frac{}{\Gamma \vdash \mathbf{false} : \mathbf{Bool}} \quad (\text{T-TRUE})$$