# 1 Algorithm 1: FixNonLocalControl

**Input:** an extracted function $EF$, an introduced function call expression $E$ (i.e., $EF(\dots)$) in the caller
**Output:** a list of patches $PS$ to apply to the refactored file

1. $PS \leftarrow []$
2. $R \leftarrow$ collect **return** statements in $EF$
3. $B, C \leftarrow$ collect top-level **break** and **continue** statements in $EF$
4. **if** $R \cup B \cup C \neq \emptyset$ **then**
   (a) $RTY \leftarrow \texttt{BuildReturnType}(R, B, C)$
   (b) $PS \leftarrow \texttt{UpdateReturnType}(EF, RTY) :: PS$
   (c) **for** $l_r \in R$ **do** $PS \leftarrow (l_r, \text{return } e \rightsquigarrow \text{return } \texttt{Ret}(e)) :: PS$
   (d) **for** $l_b \in B$ **do** $PS \leftarrow (l_b, \text{break} \rightsquigarrow \text{return } \texttt{Break}) :: PS$
   (e) **for** $l_c \in C$ **do** $PS \leftarrow (l_c, \text{continue} \rightsquigarrow \text{return } \texttt{Continue}) :: PS$
   (f) $l_E \leftarrow$ find location of the final expression of $EF$
   (g) $PS \leftarrow (l_E, E \rightsquigarrow \texttt{Ok}(E)) :: PS$
   (h) $\overline{CS} \leftarrow \texttt{BuildCasesForReturnType}(RTY)$
   (i) $l_{\text{caller}} \leftarrow$ location of $E$
   (j) $PS \leftarrow (l_{\text{caller}}, E \rightsquigarrow \text{match } E \text{ with } \overline{CS}) :: PS$
5. **return** $PS$

## 2 Algorithm 2: FixOwnershipAndBorrowing

**Input:** the extracted function $EF$, the expression $E$ of the call to $EF$, orginal function $F$
**Output:** a set of patches $PS$ (to apply to the refactored file)

1. $Aliases \leftarrow$ alias analysis on $F$
2. $Mut \leftarrow \texttt{CollectMutabilityConstraints}(EF, \texttt{Aliases})$
3. $Own \leftarrow \texttt{CollectOwnershipConstraints}(EF, \texttt{Aliases}, F)$
4. $PS \leftarrow []$
5. **for** $param \in EF.\texttt{params}$ **do**
   (a) $v, \tau, l \leftarrow param.\texttt{var}, param.\texttt{type}, param.\texttt{loc}$
   (b) **if** $\texttt{UNSAT}(\texttt{Mut} \cup \texttt{Own}, v)$ **then** raise `RefactorError`
   (c) **if** $\texttt{LUB}(\texttt{Mut} \cup \texttt{Own}, v) = \langle \texttt{mut, ref} \rangle$ **then** $PS \leftarrow (l, v : \tau \rightsquigarrow v : \&\texttt{mut}\, \tau) :: PS$
   (d) **if** $\texttt{LUB}(\texttt{Mut} \cup \texttt{Own}, v) = \langle \texttt{imm, ref} \rangle$ **then** $PS \leftarrow (l, v : \tau \rightsquigarrow v : \&\tau) :: PS$
6. **for** $param \in EF.\texttt{params}$ **do**
   (a) **if** $\texttt{LUB}(\texttt{Mut} \cup \texttt{Own}, param.\texttt{var}) = \langle \_, \texttt{ref} \rangle$ **then**
   (b) $\texttt{Exps} \leftarrow$ collect from $EF.\texttt{body}$ all the occurrences of $param.\texttt{var}$
   (c) **for** $e \in \texttt{Exps}$ **do** $PS \leftarrow (e.\texttt{loc}, e \rightsquigarrow (*e)) :: PS$
7. **for** $arg \in E.\texttt{args}$ **do**
   (a) $v, e, l \leftarrow arg.\texttt{var}, arg.\texttt{exp}, arg.\texttt{loc}$
   (b) **if** $\texttt{LUB}(\texttt{Mut} \cup \texttt{Own}, v) = \langle \texttt{mut, ref} \rangle$ **then** $PS \leftarrow (l, e \rightsquigarrow \&\texttt{mut}\, e) :: PS$
   (c) **if** $\texttt{LUB}(\texttt{Mut} \cup \texttt{Own}, v) = \langle \texttt{imm, ref} \rangle$ **then** $PS \leftarrow (l, e \rightsquigarrow \&e) :: PS$

# 3 Algorithm 3: CollectMutabilityConstraints

**Input:** extracted function $EF$, an alias map `Aliases`
**Output:** a set `Mut` of mutability constraints

1. $MV \leftarrow$ collect all the variables in $EF$ that are part of an `lvalue` expression
2. $MV \leftarrow MV +$ all variables in the body of $EF$ that are function call arguments with mutable requirements
3. $MV \leftarrow MV +$ all variables in $EF$ that are mutably borrowed
4. $\texttt{Mut} \leftarrow \{\texttt{imm} \leq p.\texttt{var} \mid p \in EF.\texttt{params} \wedge \forall v' \in \texttt{Aliases}(p.\texttt{var}) : v' \notin MV\} \cup$
5. $\quad \{\texttt{mut} \leq p.\texttt{var} \mid p \in EF.\texttt{params} \wedge \exists v' \in \texttt{Aliases}(p.\texttt{var}) : v' \in MV\}$

# 4 Algorithm 4: CollectOwnershipConstraints

**Input:** extracted function $EF$, an alias map `Aliases`, original caller function $F$
**Output:** a set `Ownership` of ownership constraints

1. $FV \leftarrow$ free variables in $F$ in the code snippet after the call to $EF$
2. $PBV \leftarrow$ collect all variables in $EF.\texttt{params}$ declared as pass-by-value
3. $\texttt{Borrows} \leftarrow PBV \cap \{p.\texttt{var} \mid p \in EF.\texttt{params} \wedge \exists v' \in \texttt{Aliases}(p.\texttt{var}) : v' \in FV\}$
4. $\texttt{Own} \leftarrow$ collect all the variables in $EF$ which are moved into or out of
5. $\texttt{Ownership} \leftarrow \{v \leq \texttt{ref} \mid v \in \texttt{Borrows}\} \cup \{\texttt{own} \leq v \mid v \in \texttt{Own}\}$

# 5 Algorithm 5: FixLifetimes

**Input:** a cargo manifest file `CARGO_MANIFEST` for the whole project, extracted function $EF$
**Output:** patched extracted function $EF'$

1. $EF' \leftarrow$ clone $EF$
2. $EF' \leftarrow$ update $EF'$ by annotating each borrow in $EF'$.`params` and $EF'$.`ret` with a fresh lifetime where none exists
3. $EF' \leftarrow$ update $EF'$ by adding the freshly introduced lifetimes to the list of lifetime parameters in $EF'$.`sig`
4. **Loop**
   (a) $err \leftarrow$ (`cargo check CARGO_MANIFEST`).`errors`
   (b) **if** $err = \emptyset$ **then break**
   (c) `suggestions` $\leftarrow$ collect lifetime bounds suggestions from $err$
   (d) **if** `suggestions` $= \emptyset$ **then** raise `RefactorError`
   (e) $EF' \leftarrow$ apply `suggestions` to $EF'$
5. // readability optimizations:
6. $EF' \leftarrow$ collapse the cycles in the `where` clause of $EF'$.`sig`
7. $EF' \leftarrow$ apply elision rules