Code Transformation for Distributed Python ML Code

1 Python Abstract Syntax

$$
\begin{array}{llll}
\mathit{module} & ::= & \mathit{stmt}^* \quad \mathit{type\_ignore} & (\textsc{ModuleDef}) \\
\mathit{stmt} & ::= & (@\mathit{expr})^* \ \texttt{def} \ \mathit{id} \ \texttt{(}\mathit{args}\texttt{)} \ \ (\texttt{->} \ \mathit{expr})? \ \ \texttt{:} \ (\texttt{\#type:}s)? \ \ \mathit{stmt}^* & (\textsc{FunDef}) \\
 & \mid & (@\mathit{expr})^* \ \texttt{async def} \ \mathit{id} \ \texttt{(}\mathit{args}\texttt{)} \ \ (\texttt{->} \ \mathit{expr})? \ \ \texttt{:} \ (\texttt{\#type:}s)? \ \ \mathit{stmt}^* & (\textsc{AsyncFunDef}) \\
 & \mid & (@\mathit{expr})^* \ \texttt{class} \ \mathit{id} \ \texttt{(}\mathit{expr}^* \ \mathit{keyword}^*\texttt{)} \ \ \texttt{:} \ \ \mathit{stmt}^* & (\textsc{ClassDef}) \\
 & \mid & \texttt{return} \ \mathit{expr}? & (\textsc{Return}) \\
 & \mid & \texttt{delete} \ \mathit{expr}^* & (\textsc{Delete}) \\
 & \mid & \mathit{expr}^* \ \ \texttt{=} \ \ \mathit{expr} \ \ (\texttt{\#type:}s)? & (\textsc{Assign}) \\
 & \mid & \mathit{expr} \ \ \mathit{binop} \ \texttt{=} \ \ \mathit{expr} & (\textsc{AugAssign}) \\
 & \mid & \mathit{expr} \ \ \texttt{:} \ \ \mathit{expr} \ \ (\texttt{=} \ \mathit{expr})? & (\textsc{AnnAssign}) \\
 & \mid & (\texttt{\#type:}s)? \ \ \texttt{for} \ \mathit{expr} \ \texttt{in} \ \mathit{expr} \ \ \texttt{:} \ \ \mathit{stmt}^* \ \ (\texttt{else} \ \texttt{:} \ \ \mathit{stmt}^*)? & (\textsc{ForLoop}) \\
 & \mid & (\texttt{\#type:}s)? \ \ \texttt{async for} \ \mathit{expr} \ \texttt{in} \ \mathit{expr} \ \ \texttt{:} \ \ \mathit{stmt}^* \ \ (\texttt{else} \ \texttt{:} \ \ \mathit{stmt}^*)? & (\textsc{AsyncForLoop}) \\
 & \mid & \texttt{while} \ \texttt{(}\mathit{expr}\texttt{)} \ \ \texttt{:} \ \ \mathit{stmt}^* \ \ (\texttt{else} \ \texttt{:} \ \ \mathit{stmt}^*)? & (\textsc{WhileLoop}) \\
 & \mid & \texttt{if} \ \texttt{(}\mathit{expr}\texttt{)} \ \ \texttt{:} \ \ \mathit{stmt}^* \ \ (\texttt{else} \ \texttt{:} \ \ \mathit{stmt}^*)? & (\textsc{If}) \\
 & \mid & (\texttt{\#type:}s)? \ \ \texttt{with} \ \mathit{with\_item}^* \ \ \texttt{:} \ \ \mathit{stmt}^* & (\textsc{With}) \\
 & \mid & (\texttt{\#type:}s)? \ \ \texttt{async with} \ \mathit{with\_item}^* \ \ \texttt{:} \ \ \mathit{stmt}^* & (\textsc{AsyncWith}) \\
 & \mid & \texttt{match} \ \mathit{expr} \ \ \texttt{:} \ \ \mathit{match\_case}^* & (\textsc{Match}) \\
 & \mid & \texttt{raise} \ \mathit{expr}? \ \ (\texttt{from} \ \mathit{expr})? & (\textsc{Raise}) \\
 & \mid & \texttt{try} \ \ \texttt{:} \ \ \mathit{stmt}^* \ \ \mathit{exc\_handler}^* \ \ (\texttt{else} \ \texttt{:} \ \ \mathit{stmt}^*)? \ \ (\texttt{finally} \ \texttt{:} \ \ \mathit{stmt}^*)? & (\textsc{Try}) \\
 & \mid & \texttt{assert} \ \mathit{expr} \ \ \mathit{expr}? & (\textsc{Assert}) \\
 & \mid & \texttt{import} \ \mathit{alias}^* & (\textsc{Import}) \\
 & \mid & \texttt{from} \ \ i \ \ \mathit{id}? \ \texttt{import} \ \mathit{alias}^* & (\textsc{ImportFrom}) \\
 & \mid & \texttt{global} \ \mathit{id}^* & (\textsc{Global}) \\
 & \mid & \texttt{nonlocal} \ \mathit{id}^* & (\textsc{NonLocal}) \\
 & \mid & \mathit{expr} & (\textsc{ExprStmt}) \\
 & \mid & \texttt{pass} & (\textsc{Pass}) \\
 & \mid & \texttt{break} & (\textsc{Break}) \\
 & \mid & \texttt{continue} & (\textsc{Continue})
\end{array}
$$

$$
\begin{array}{llll}
\mathit{expr} & ::= & \mathit{expr}\ \ \mathit{boolop}\ \ \mathit{expr} & (\textsc{BoolOp}) \\
 & \mid & \mathit{expr}\ \ \texttt{:=}\ \ \mathit{expr} & (\textsc{NamedExpr}) \\
 & \mid & \mathit{expr}\ \ \mathit{binop}\ \ \mathit{expr} & (\textsc{BinaryOp}) \\
 & \mid & \mathit{unop}\ \ \mathit{expr} & (\textsc{UnaryOp}) \\
 & \mid & \texttt{lambda}\ \mathit{args}\ \ \texttt{:}\ \ \mathit{expr} & (\textsc{Lambda}) \\
 & \mid & \mathit{expr}\ \ \texttt{if}\ \ \mathit{expr}\ \ \texttt{else}\ \ \mathit{expr} & (\textsc{IfExpr}) \\
 & \mid & \{(\mathit{expr}\ \texttt{:}\ \mathit{expr})^*\ \ (**\mathit{expr})^*\} & (\textsc{Dictionary}) \\
 & \mid & \{\mathit{expr}^*\} & (\textsc{Set}) \\
 & \mid & \texttt{[}\mathit{expr}^*\texttt{]} & (\textsc{List}) \\
 & \mid & \texttt{(}\mathit{expr}^*\texttt{)} & (\textsc{Tuple}) \\
 & \mid & \{\mathit{expr}\texttt{:}\mathit{expr}\ \ \ \mathit{comprehension}^*\} & (\textsc{DictionaryComp} \\
 & \mid & \{\mathit{expr}\ \ \ \mathit{comprehension}^*\} & (\textsc{SetComp}) \\
 & \mid & \texttt{[}\mathit{expr}\ \ \ \mathit{comprehension}^*\texttt{]} & (\textsc{ListComp}) \\
 & \mid & \texttt{(}\mathit{expr}\ \ \mathit{comprehension}^*\texttt{)} & (\textsc{GeneratorComp} \\
 & \mid & \texttt{await}\ \ \mathit{expr} & (\textsc{Await}) \\
 & \mid & \texttt{yield}\ \ \mathit{expr}? & (\textsc{Yield}) \\
 & \mid & \texttt{yield from}\ \ \mathit{expr} & (\textsc{YieldFrom}) \\
 & \mid & \mathit{expr}\ \ (\mathit{compop}\ \mathit{expr})^* & (\textsc{CompOp}) \\
 & \mid & \mathit{expr}\ \ \texttt{(}\mathit{expr}^*\ \mathit{keyword}^*\texttt{)} & (\textsc{Call}) \\
 & \mid & \{\mathit{expr}\ \ \ \texttt{(!}\mathit{i}\texttt{)?}\ \ \texttt{(:}\mathit{expr}\texttt{)?}\} & (\textsc{FormattedValue} \\
 & \mid & \mathit{expr}^* & (\textsc{JoinedStr}) \\
 & \mid & \mathit{constant} & (\textsc{Constant}) \\
 & \mid & \mathit{expr}\texttt{.}\mathit{id} & (\textsc{Attribute}) \\
 & \mid & \mathit{expr}\texttt{[}\mathit{expr}\texttt{]} & (\textsc{Subscript}) \\
 & \mid & \texttt{*}\mathit{expr} & (\textsc{Starred}) \\
 & \mid & \texttt{**}\mathit{expr} & (\textsc{DoubleStarred}) \\
 & \mid & \mathit{id} & (\textsc{Name}) \\
 & \mid & \mathit{expr}?\ (\texttt{:}\mathit{expr})?\ (\texttt{:}\mathit{expr})? & (\textsc{Slice}) \\
\mathit{boolop} & ::= & \texttt{and} \mid \texttt{or} & (\textsc{BoolOperator}) \\
\mathit{binop} & ::= & \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{@} \mid \texttt{/} \mid \texttt{**} \mid \texttt{<<} \mid \texttt{>>} \mid \texttt{|} \mid \texttt{\^{}} \mid \texttt{\&} \mid \texttt{//} \mid \texttt{\%} & (\textsc{BinOperator}) \\
\mathit{unop} & ::= & \sim \mid \texttt{not} \mid \texttt{+} \mid \texttt{-} & (\textsc{UnOperator}) \\
\mathit{compop} & ::= & \texttt{==} \mid \texttt{!=} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{>=} \mid \texttt{is} \mid \texttt{is not} \mid \texttt{in} \mid \texttt{not in} & (\textsc{CompOperator}) \\
\mathit{comprehension} & ::= & \texttt{for}\ \ \mathit{expr}\ \ \texttt{in}\ \ \mathit{expr}\ \ (\texttt{if}\ \mathit{expr})^* & (\textsc{Comprehension}) \\
 & \mid & \texttt{async for}\ \ \mathit{expr}\ \ \texttt{in}\ \ \mathit{expr}\ \ (\texttt{if}\ \mathit{expr})^* & (\textsc{AsyncComprehen} \\
\mathit{exc\_handler} & ::= & \texttt{except}\ \ \mathit{expr}?\ (\texttt{as}\ \ \mathit{id})?\ \texttt{:}\ \ \mathit{stmt}^* & (\textsc{ExcHandler}) \\
\mathit{args} & ::= & (\mathit{arg}\ (=\mathit{expr})?)^*,\ \ (\mathit{arg}\ (=\mathit{expr})?)^*,\ \ \mathit{arg}?,\ \ (\mathit{arg}\ (=\mathit{expr})?)^*,\ \ \mathit{arg}? & (\textsc{Arguments}) \\
\mathit{arg} & ::= & \mathit{id}\ \ \mathit{expr}?\ \mathit{s}? & (\textsc{Argument}) \\
\mathit{keyword} & ::= & \mathit{id}?\ \texttt{=}\ \mathit{expr} & (\textsc{Keyword}) \\
\mathit{alias} & ::= & \mathit{id}\ (.\mathit{id})^*\ (\texttt{as}\ \ \mathit{id})? & (\textsc{Alias}) \\
\mathit{with\_item} & ::= & \mathit{expr}\ \ (\texttt{as}\ \ \mathit{expr})? & (\textsc{WithItem})
\end{array}
$$

$$
\begin{array}{lcll}
\mathit{match\_case} & ::= & \texttt{case}\ \mathit{pattern}\ (\texttt{if}\ \mathit{expr})?\ \texttt{:}\ \ \mathit{stmt}^* & (\textsc{MatchCase}) \\
\mathit{pattern} & ::= & \mathit{expr} & (\textsc{MatchValue}) \\
 & | & \mathit{constant} & (\textsc{MatchSingleton}) \\
 & | & \texttt{[}\mathit{pattern}^*\texttt{]} & (\textsc{MatchSequence}) \\
 & | & \texttt{*}(\mathit{id})? & (\textsc{MatchStar}) \\
 & | & \{(\mathit{expr}\ \texttt{:}\ \mathit{pattern})^*\ \ \mathit{id}?\} & (\textsc{MatchMapping}) \\
 & | & \mathit{expr}\ \ \texttt{(}\mathit{pattern}^*\ (\mathit{id} = \mathit{pattern})^*\texttt{)} & (\textsc{MatchClass}) \\
 & | & (\mathit{pattern}\ \ \texttt{as})?\ \mathit{id} & (\textsc{MatchAs}) \\
 & | & \mathit{pattern}\ \ \texttt{|}\ \mathit{pattern} & (\textsc{MatchOr}) \\
 & | & \texttt{\_} & (\textsc{MatchWildcard}) \\
\mathit{constant} & ::= & \texttt{None} & (\textsc{NoneLiteral}) \\
 & | & i & (\textsc{IntLiteral}) \\
 & | & f & (\textsc{FloatLiteral}) \\
 & | & c & (\textsc{ComplexLiteral}) \\
 & | & s & (\textsc{StringLiteral}) \\
 & | & b & (\textsc{BooleanLiteral}) \\
 & | & \texttt{(}\mathit{constant}^*\texttt{)} & (\textsc{TupleLiteral}) \\
 & | & \texttt{...} & (\textsc{Ellipsis}) \\
\mathit{type\_ignore} & ::= & i^* & (\textsc{TypeIgnore}) \\
\mathit{id} & \in & \mathit{Id} & \\
s & \in & \mathit{Str} & \\
b & \in & \{\texttt{True}, \texttt{False}\} & \\
i & \in & \mathbb{Z} & \\
f & \in & \mathbb{R} & \\
c & \in & \mathbb{C} &
\end{array}
$$

Note: there may be more constant terms such as an immutable container containing only constant elements. Please update the *constant* rule if you know such terms.

2 Transformation for TF2 Python Code

2.1 Restrictions

1. All import statements must be placed at the top of a module.
2. The `tensorflow` module must be assigned to a variable only using an import statement.
3. Members of the `tensorflow` module must not be aliased.
4. `print` function call expressions must not introduce side-effects.
5. A dataset and an optimizer object must only be created once via an assignment statement with a function call expression and must not be aliased.
6. Variables storing a dataset or a optimizer object must not be reassigned to store another value that is not a dataset or an optimizer object.
7. A dataset and an optimizer object must not be created conditionally.
8. `optimizer.apply_gradients` function call expressions must be expression statements or a direct RHS of an assignment statement.
9. Global variables storing an `optimizer` object and referred in functions must be defined before the functions' definitions and must not be changed after their initializations.
10. A checkpoint object must only be created once via an assignment statement with a function call expression and must not be aliased.

2.2 Rules

2.2.1 Types and Auxiliary Functions

$$
\begin{array}{lll}
\tau & ::= & \mathit{Module} & (\textsc{Modules}) \\
& & \mathit{Stmt} & (\textsc{Statements}) \\
& & \mathit{Expr} & (\textsc{Expressions}) \\
& & \mathit{Comprehension} & (\textsc{Comprehensions}) \\
& & \mathit{ExcHandler} & (\textsc{Exception handlers}) \\
& & \mathit{Alias} & (\textsc{Aliases}) \\
& & \mathit{WithItem} & (\textsc{WithItems}) \\
& & \mathit{Pattern} & (\textsc{Patterns}) \\
& & \mathit{Id} & (\textsc{Identifiers}) \\
& & \mathit{Str} & (\textsc{Strings}) \\
& & \tau \text{ list} & (\textsc{List of elements typed } \tau)
\end{array}
$$

$\sigma \in \Sigma = \mathit{Str} \xrightarrow{\text{fin}} \mathit{Id}$ $\quad$ $\textsc{Environment storing mappings from strings to identifiers}$

$\mathtt{ClassNode} = \mathit{Str} \times \mathtt{ClassNode}?$

$\mathtt{ns} \in \mathtt{Nodes} ::= \mathtt{ClassNode}\ \text{list}$

$\mathtt{vs} \in \mathtt{Vars} ::= \mathit{Str}\ \text{list} \xrightarrow{\text{fin}} \mathtt{ClassNode}$

$\mathtt{cg} \in \mathtt{CG} = \mathtt{Nodes} \times \mathtt{Vars}$

._1	:	$\tau \times \tau \rightarrow \tau$	GET THE FIRST ELEMENT OF THE GIVEN PAIR
@	:	τ list $\rightarrow \tau$ list $\rightarrow \tau$ list	CONCATENATE TWO LISTS
::	:	$\tau \rightarrow \tau$ list $\rightarrow \tau$ list	APPEND AN ELEMENT TO A LIST (RIGHT-ASSOCIATIVE)
.nodes	:	`ClassNode`$\rightarrow$ `Nodes` = **._1**	GET NODES OF CLASS GRAPH
.vars	:	`ClassNode`$\rightarrow$ `Vars` = **._2**	GET VARIABLES OF CLASS GRAPH

$\boxed{\textbf{getPath: } \textit{Expr} \ \rightarrow \ \texttt{CG} \ \rightarrow \ (\textit{Str} \ \textbf{list} \ \times \ \texttt{CG} \ \times \ \texttt{ClassNode})}$

$\textbf{getPath}[\![\ \textit{expr}.\ \textit{id}\ ]\!](\texttt{cg}) = \textbf{LET} \ \ (\texttt{ns}, \texttt{vs}) \ \ = \ \texttt{cg} \ \ \textbf{IN}$

$\quad \textbf{LET} \ \ (l, \texttt{ns}_1, \textit{node}_1) \ \ = \ \textbf{getPath}[\![\ \textit{expr}\ ]\!](\texttt{cg}) \ \textbf{IN}$

$\quad\quad \textbf{IF} \ \ \textbf{isChild}(\texttt{cg})(\textit{node}_1, \textit{id}) \ \ = \ \textit{node}_2 \ \ \textbf{THEN} \ \ (l{+}\mathrm{str}(\textit{id}), \texttt{ns}_1, \textit{node}_2)$

$\quad\quad \textbf{ELSE} \ \textbf{LET} \ \ \textit{node}_2 \ \ = \ \{\ \textit{id}, \textit{node}_1\ \} \ \ \textbf{IN} \ \ (l{+}\mathrm{str}(\textit{id}), \texttt{ns}_1, \textit{node}_2)$

$\textbf{getPath}[\![\ \textit{id}\ ]\!](\texttt{cg}) = \textbf{LET} \ \ (\texttt{ns}, \texttt{vs}) \ \ = \ \texttt{cg} \ \ \textbf{IN}$

$\quad \textbf{IF} \ \ \textit{id} \ \in \mathrm{Dom}(\texttt{vs}) \ \ \textbf{THEN}$

$\quad\quad \textbf{LET} \ \ \textit{node}_1 \ \ = \ \texttt{vs}(\textit{id}) \ \ \textbf{IN} \ \ (\mathrm{str}(\textit{id}), \texttt{ns}, \textit{node}_1)$

$\quad \textbf{ELSE} \ \ \textbf{LET} \ \ \textit{node}_1 \ \ = \ \{\ \textit{id}, \cdot\ \} \ \ \textbf{IN} \ \ (\mathrm{str}(\textit{id}), \texttt{ns}\textbf{@}[\textit{node}_1], \textit{node}_1)$

$\boxed{\textbf{isChild: } \texttt{CG} \ \rightarrow \ (\texttt{ClassNode} \ \times \ \textit{Id}) \ \rightarrow \ \texttt{ClassNode?}}$

$\textbf{isChild}(\texttt{cg})(\textit{node}_t, \textit{id}_t) \ \ =$

$\quad \texttt{cg}\textbf{.nodes}\texttt{.filter}(\textit{node} \Rightarrow \textit{node}\textbf{.\_1} \ = \ \textit{id}_t \ \ \wedge \ \ \textit{node}\textbf{.\_2} \ = \ \textit{node}_t)\texttt{.headOpt}$

2.2.2 Transformation Rules

$\boxed{\textit{trans}_M \ \ \textbf{:} \ \ \textit{Module} \ \rightarrow \ \textit{Module}}$

$\textit{trans}_M[\![\ \textit{stmt}^* \ \ \textit{type\_ignore}\ ]\!] \ \ = \ \ \textit{trans}_{\overline{S}}[\![\ \textit{stmt}^*\ ]\!](\sigma)\textbf{.\_1} \ \ \textit{type\_ignore}$

$\boxed{\textit{trans}_{\overline{S}} \ \ \textbf{:} \ \ \textit{Stmt} \ \textbf{list} \ \ \rightarrow \ \ \Sigma \ \rightarrow \ \ (\textit{Stmt} \ \textbf{list} \ \ \times \ \ \Sigma)}$

$$\begin{aligned}\textit{trans}_{\overline{S}}[\![\ \textit{stmt}_1 \ \ \textit{stmt}_2 \ ... \ \textit{stmt}_n\ ]\!](\sigma) \ \ = \ \ & \textbf{LET} \ \ \textit{stmt}_1^{*\prime}, \sigma_1 \ \ = \ \textit{trans}_S[\![\ \textit{stmt}_1\ ]\!](\sigma) \ \ \textbf{IN} \\ & \textbf{LET} \ \ \textit{stmt}_2^{*\prime}, \sigma_2 \ \ = \ \textit{trans}_S[\![\ \textit{stmt}_2\ ]\!](\sigma_1) \ \ \textbf{IN} \\ & \ldots \\ & \textbf{LET} \ \ \textit{stmt}_n^{*\prime}, \sigma_n \ \ = \ \textit{trans}_S[\![\ \textit{stmt}_n\ ]\!](\sigma_{n-1}) \ \ \textbf{IN} \\ & (\textit{stmt}_1^{*\prime} \ \textbf{@} \ \textit{stmt}_2^{*\prime} \ \textbf{@} \ ... \ \textbf{@} \ \textit{stmt}_n^{*\prime}, \sigma_n)\end{aligned}$$

$\boxed{\textit{trans}_S \ \ \textbf{:} \ \ \textit{Stmt} \ \rightarrow \ \ \texttt{CG} \ \rightarrow \ \ (\textit{Stmt} \ \textbf{list} \ \ \times \ \ \texttt{CG})}$

$\textit{trans}_S[\![\ (@\textit{expr}_1)^* \ \ \texttt{def} \ \textit{id} \ \ (\textit{args}) \ \ (\texttt{->} \ \textit{expr}_2)? \ \ \ \texttt{:} \ \ (\texttt{\#type:}s)? \ \ \textit{stmt}^*\ ]\!](\sigma) =$

$\quad ([(@\textit{expr}_1)^* \ \ \texttt{def} \ \textit{id} \ \ (\textit{args}) \ \ (\texttt{->} \ \textit{expr}_2)? \ \ \ \texttt{:} \ \ (\texttt{\#type:}s)? \ \ \textit{trans}_{\overline{S}}[\![\ \textit{stmt}^*\ ]\!](\sigma)\textbf{.\_1}], \sigma)$

$\textit{trans}_S[\![\ (@\textit{expr}_1)^* \ \ \texttt{async def} \ \textit{id} \ \ (\textit{args}) \ \ (\texttt{->} \ \textit{expr}_2)? \ \ \ \texttt{:} \ \ (\texttt{\#type:}s)? \ \ \textit{stmt}^*\ ]\!](\sigma) =$

$\quad ([(@\textit{expr}_1)^* \ \ \texttt{async def} \ \textit{id} \ \ (\textit{args}) \ \ (\texttt{->} \ \textit{expr}_2)? \ \ \ \texttt{:} \ \ (\texttt{\#type:}s)? \ \ \textit{trans}_{\overline{S}}[\![\ \textit{stmt}^*\ ]\!](\sigma)\textbf{.\_1}], \sigma)$

$\textit{trans}_S[\![\ (@\textit{expr}_1)^* \ \ \texttt{class} \ \textit{id} \ \ (\textit{expr}_2^* \ \textit{keyword}^*) \ \ \ \texttt{:} \ \ \textit{stmt}^*\ ]\!](\sigma) =$

$\quad ([(@\textit{expr}_1)^* \ \ \texttt{class} \ \textit{id} \ \ (\textit{expr}_2^* \ \textit{keyword}^*) \ \ \ \texttt{:} \ \ \textit{trans}_{\overline{S}}[\![\ \textit{stmt}^*\ ]\!](\sigma)\textbf{.\_1}], \sigma) =$

$\textit{trans}_S[\![\ \texttt{return} \ \ \textit{expr}?\ ]\!](\sigma) = ([\texttt{return} \ \ (\textit{trans}_E[\![\ \textit{expr}\ ]\!](\sigma))?], \sigma)$

$trans_S⟦$ `delete` $expr^*$ $⟧(\sigma) = ([$`delete` $expr^*], \sigma)$

A strict form of assignment statements

$trans_S⟦\ id_r$ `=` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?$ $⟧($`cg`$) =$

LET $($`ns`$,$ `vs`$) =$ `cg` **IN**

  LET **getPath**$(expr_1)($`cg`$) = ($`tensorflow.optimizers.Adam`$,$ `ns`$_1, node_1)$ **THEN**

   IF id_i $=$ `learning_rate` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**

    $([id_r$ `=` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... id_i `=` $expr_{2i}$ `* hvd.size()`

     ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?]$, $($`ns`$_1$, `vs`$[id_r \mapsto node_1])$ $)$

   ELSE

    $([id_r$ `=` $expr_1$ `(`$expr_{11}$ `* hvd.size()` ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ...

     $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?]$, $($`ns`$_1$, `vs`$[id_r \mapsto node_1])$ $)$

  IF $\sigma($**"tensor_flow"**$) = id_t$ **AND** $expr_1 = id_t$`.data.Dataset.`$expr_3$ **THEN**

   $([id_r$ `=` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?]$,

    $\sigma[$**"dataset"**$\mapsto id_r])$

  ELIF $\sigma($**"tensor_flow"**$) = id_t$ **AND** $expr_1 = id_t$`.train.Checkpoint` **THEN**

   $([id_r$ `=` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?]$,

    $\sigma[$**"checkpoint"**$\mapsto id_r])$

  ELIF $\sigma($**"tensor_flow"**$) = id_t$ **AND** $expr_1 = id_t$`.optimizers.Adam` **THEN**

   IF id_i $=$ `learning_rate` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**

    $([id_r$ `=` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... id_i `=` $expr_{2i}$ `* hvd.size()`

     ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?]$, $\sigma[$**"optimizer"**$\mapsto id_r])$

   ELSE

    $([id_r$ `=` $expr_1$ `(`$expr_{11}$ `* hvd.size()` ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?]$,

     $\sigma[$**"optimizer"**$\mapsto id_r])$

  ELIF $\sigma($**"optimizers"**$) = id_t$ **AND** $expr_1 = id_t$`.Adam` **THEN**

   IF id_i $=$ `learning_rate` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**

    $([id_r$ `=` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... id_i `=` $expr_{2i}$ `* hvd.size()`

     ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?]$, $\sigma[$**"optimizer"**$\mapsto id_r])$

   ELSE

    $([id_r$ `=` $expr_1$ `(`$expr_{11}$ `* hvd.size()` ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?]$,

     $\sigma[$**"optimizer"**$\mapsto id_r])$

  ELIF $\sigma($**"optimizer"**$) = id_t$ **AND** $expr_1 = id_t$`.apply_gradients` **THEN**

   IF id_i $=$ `grads_and_vars` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**

    LET id_z $=$ **NewID()** **IN**

    $([id_z$ `=` $expr_{2i}$,

    id_r `=` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ $(id_1$ `=` $)?$ $expr_{21}$ ... id_i `=` id_z ... $(id_k$ `=` $)?$ $expr_{2k}$`)` $($`#type:`$s)?$,

    `global hvd_broadcast_done`,

    `if not hvd_broadcast_done:` $[$ `hvd.broadcast_variables([x[1] for x in` id_z`], root_rank=0)`,

     `hvd.broadcast_variables(`id_t`.variables(), root_rank=0)`,

     `hvd_broadcast_done = True` $]]$, $\sigma)$

   ELSE

    LET id_z $=$ **NewID()** **IN**

$([id_z$ = $expr_{11},$
id_r = $expr_1$ (id_z $expr_{12}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$) (#type:s)?,
global hvd_broadcast_done,
if not hvd_broadcast_done: [hvd.broadcast_variables([x[1] for x in id_z], root_rank=0),
hvd.broadcast_variables(id_t.variables(), root_rank=0),
hvd_broadcast_done = True]], σ)

ELIF σ(**"checkpoint"**) $=$ id_t **AND** $expr_1$ $=$ id_t.save **THEN**
([if hvd.rank() == 0: [id_r = $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$) (#type:s)?]],
σ)

ELIF σ(**"keras"**) $=$ id_k **AND** $expr_1$ $=$ id_k.models.Sequential **THEN**
([id_r = $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_2 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$)], σ[**"model"**$\mapsto$ id_r])

ELIF σ(**"keras"**) $=$ id_t **AND** $expr_1$ $=$ id_t.optimizers.Adam **THEN**
IF id_i $=$ learning_rate **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**
([id_r = $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... id_i = $expr_{2i}$ * hvd.size()
... (id_k =)? $expr_{2k}$) (#type:s)?
id_r = hvd.DistributedOptimizer(id_r)], σ[**"optimizer"**$\mapsto$ id_r])
ELSE
([id_r = $expr_1$ ($expr_{11}$ * hvd.size() ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$) (#type:s)?
id_r = hvd.DistributedOptimizer(id_r)], σ[**"optimizer"**$\mapsto$ id_r])

ELSE ([id_r = $trans_E[\![$ $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_2 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$) $]\!](\sigma)$ (#type:s)?],
σ)

$trans_S[\![$ $expr_0$ = $expr_1$ (#type:s)? $]\!](\sigma) =$
IF σ(**"os"**) $=$ id_{os} **AND** $expr_0$ $=$ id_{os}.environ ['CUDA_VISIBLE_DEVICES']
THEN ([], σ)

A general form of assignment statements except for the strict form

$trans_S[\![$ $expr_1^*$ = $expr_2$ (#type:s)? $]\!](\sigma) =$ ([$expr_1^*$ = $trans_E[\![$ $expr_2$ $]\!](\sigma)$ (#type:s)?], σ)

$trans_S[\![$ $expr_1$ $binop$ = $expr_2$ $]\!](\sigma) =$ ([$expr_1$ $binop$ = $trans_E[\![$ $expr_2$ $]\!](\sigma)$], σ)

$trans_S[\![$ $expr_1$: $expr_2$ (= $expr_3$)? $]\!](\sigma) =$ ([$expr_1$: $expr_2$ (= $trans_E[\![$ $expr_3$ $]\!](\sigma)$)?], σ)
IF $expr_1$ $=$ id_1 **AND** σ(**"tensor_flow"**) $=$ id_2
AND $expr_3$ $=$ id_2.data.Dataset.$expr_4$ ($expr_5^*$ $keyword^*$) **THEN**
([$expr_1$: $expr_2$ = $expr_3$ (#type:s)?], σ[**"dataset"**$\mapsto$ id_1])
ELSE ([$expr_1$: $expr_2$ (= $trans_E[\![$ $expr_3$ $]\!](\sigma)$)?], σ)

$trans_S[\![$ (#type:s)? for $expr_1$ in $expr_2$: $stmt_1^*$ (else : $stmt_2^*$)? $]\!](\sigma) =$
([(#type:s)? for $expr_1$ in $trans_E[\![$ $expr_2$ $]\!](\sigma)$:
$trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)$._**1** (else : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma)$._**1**)?], σ)

$trans_S[\![$ (#type:s)? async for $expr_1$ in $expr_2$: $stmt_1^*$ (else : $stmt_2^*$)? $]\!](\sigma) =$
([(#type:s)? async for $expr_1$ in $trans_E[\![$ $expr_2$ $]\!](\sigma)$:
$trans_{\overline{S}}[\![$ $stmt_1^*$ $]\!](\sigma)$._**1** (else : $trans_{\overline{S}}[\![$ $stmt_2^*$ $]\!](\sigma)$._**1**)?], σ)

$trans_S[\![$ `while (`$expr$`) :` $stmt_1^*$ `(else :` $stmt_2^*$`)?` $]\!](\sigma) =$
$([$`while (`$trans_E[\![\ expr\ ]\!](\sigma)$`) :` $trans_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma).\_\mathbf{1}$ `(else :` $trans_{\overline{S}}[\![\ stmt_2^*\ ]\!](\sigma).\_\mathbf{1}$`)?`$], \sigma)$

$trans_S[\![$ `if (`$expr$`) :` $stmt^*$ `(else :` $stmt^*$`)?` $]\!](\sigma) =$
$([$`if (`$trans_E[\![\ expr\ ]\!](\sigma)$`) :` $trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma).\_\mathbf{1}$ `(else :` $trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma).\_\mathbf{1}$`)?`$], \sigma)$

$trans_S[\![$ `(#type:`s`)? with` $with\_item^*$ `:` $stmt^*$ $]\!](\sigma) =$
LET $with\_item^{*\prime}, \sigma_1 = trans_{\overline{W}}[\![\ with\_item^*\ ]\!](\sigma)$ **IN**
LET $stmt^{*\prime}, \sigma_2 = trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma_1)$ **IN**
IF $\sigma_1 \setminus \sigma = [$**"gradient_tape"**$\mapsto id]$ **THEN**
$([$`(#type:`s`)? with` $with\_item^{*\prime}$ `:` $stmt^{*\prime}$,
id `= hvd.DistributedGradientTape(`id`)`$], \sigma_2)$
ELSE $([$`(#type:`s`)? with` $with\_item^{*\prime}$ `:` $stmt^{*\prime}$$], \sigma_2)$

$trans_S[\![$ `(#type:`s`)? async with` $with\_item^*$ `:` $stmt^*$ $]\!](\sigma) =$
LET $with\_item^{*\prime}, \sigma_1 = trans_{\overline{W}}[\![\ with\_item^*\ ]\!](\sigma)$ **IN**
LET $stmt^{*\prime}, \sigma_2 = trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma_1)$ **IN**
IF $\sigma_1 \setminus \sigma = [$**"gradient_tape"**$\mapsto id]$ **THEN**
$([$`(#type:`s`)? async with` $with\_item^{*\prime}$ `:` $stmt^{*\prime}$,
id `= hvd.DistributedGradientTape(`id`)`$], \sigma_2)$
ELSE $([$`(#type:`s`)? async with` $with\_item^{*\prime}$ `:` $stmt^{*\prime}$$], \sigma_2)$

$trans_S[\![$ `match` $expr$ `:` $match\_case^*$ $]\!](\sigma) =$
$([$`match` $trans_E[\![\ expr\ ]\!](\sigma)$ `:` $(trans_C[\![match\_case]\!](\sigma))^*], \sigma)$

$trans_S[\![$ `raise` $expr_1?$ `(from` $expr_2$`)?` $]\!](\sigma) = ([$`raise` $expr_1?$ `(from` $expr_2$`)?`$], \sigma)$

$trans_S[\![$ `try :` $stmt_1^*$ $exc\_handler^*$ `(else :` $stmt_2^*$`)?` `(finally :` $stmt_3^*$`)?` $]\!](\sigma) =$
$([$`try :` $trans_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma).\_\mathbf{1}$ $(trans_H[\![exc\_handler]\!](\sigma))^*$
`(else :` $trans_{\overline{S}}[\![\ stmt_2^*\ ]\!](\sigma).\_\mathbf{1}$`)?` `(finally :` $trans_{\overline{S}}[\![\ stmt_3^*\ ]\!](\sigma).\_\mathbf{1}$`)?`$], \sigma)$

$trans_S[\![$ `assert` $expr_1$ $expr_2?$ $]\!](\sigma) = ([$`assert` $trans_E[\![\ expr_1\ ]\!](\sigma)$ $expr_2?], \sigma)$

$trans_S[\![$ `import` $alias^*$ $]\!](\mathtt{cg}) =$ **LET** $(\mathtt{ns}, \mathtt{vs}) = \mathtt{cg}$ **IN**
LET $(\mathtt{ns}_1, \mathtt{vs}_1) = trans_{\overline{A}}[\![\ alias^*\ ]\!](\mathtt{cg})$ **IN**
IF $\mathtt{vs}_1 \setminus \mathtt{vs} = [id_t \mapsto \{$ **"tensor_flow"**, $node_t\ \}]$ **THEN**
$([$`import` $alias^*$,
`import horovod.tensorflow as hvd`,
`hvd_broadcast_done = False`,
`hvd.init()`,
`gpus = `id_t`.config.experimental.list_physical_devices('GPU')`,
`for gpu in gpus: `id_t`.config.experimental.set_memory_growth(gpu, True)`,
`if gpus: `id_t`.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')`$], (\mathtt{ns}_1, \mathtt{vs}_1))$
ELSE $([$`import` $alias^*], \mathtt{cg})$

$trans_S[\![$ `from` i $id?$ `import` $alias^*$ $]\!](\sigma) =$

LET $\sigma_1 = trans_{\overline{A}}[\![$ $alias^*$ $]\!](\sigma)$ **IN**

IF $id =$ `tensorflow` **AND** $\sigma_1 \setminus \sigma =$ [**"keras"**$\mapsto id$] **THEN**

([`from` i $id?$ `import` $alias^*$,

`import horovod.tensorflow.keras as hvd`,

`hvd.init()`,

`gpus =` id`.config.experimental.list_physical_devices('GPU')`,

`for gpu in gpus:` id`.config.experimental.set_memory_growth(gpu, True)`,

`if gpus:` id`.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')`], σ_1)

ELSE ([`from` i $id?$ `import` $alias^*$], σ_1)

$trans_S[\![$ `global` id^* $]\!](\sigma) =$ ([`global` id^*], σ)

$trans_S[\![$ `nonlocal` id^* $]\!](\sigma) =$ ([`nonlocal` id^*], σ)

A strict form of expr statements

$trans_S[\![$ $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ (id_1 `=`)? $expr_{21}$ ... (id_k `=`)? $expr_{2k}$`)` $]\!](\sigma) =$

IF σ(**"optimizer"**) $= id_t$ **AND** $expr_1 = id_t$`.apply_gradients` **THEN**

IF $id_i =$ `grads_and_vars` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**

LET id_z = **NewID()** **IN**

([id_z `=` $expr_{2i}$,

$expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ (id_1 `=`)? $expr_{21}$ ... id_i `=` id_z ... (id_k `=`)? $expr_{2k}$`)`,

`global hvd_broadcast_done`,

`if not hvd_broadcast_done:` [`hvd.broadcast_variables([x[1] for x in` id_z`], root_rank=0)`,

`hvd.broadcast_variables(`id_t`.variables(), root_rank=0)`,

`hvd_broadcast_done = True`]], σ)

ELSE

LET id_z = **NewID()** **IN**

([id_z `=` $expr_{11}$,

$expr_1$ `(`id_z $expr_{12}$ ... $expr_{1n}$ (id_1 `=`)? $expr_{21}$ ... (id_k `=`)? $expr_{2k}$`)`,

`global hvd_broadcast_done`,

`if not hvd_broadcast_done:` [`hvd.broadcast_variables([x[1] for x in` id_z`], root_rank=0)`,

`hvd.broadcast_variables(`id_t`.variables(), root_rank=0)`,

`hvd_broadcast_done = True`]], σ)

ELIF $expr_1 =$ `print` **OR** $expr_1 = expr$`.write` **OR** $expr_1 = expr$`.summary` **OR**

$expr_1 = expr$`.save_weights` **OR** $expr_1 = expr$`.load_weights` **THEN**

([`if hvd.rank() == 0:` $expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ (id_1 `=`)? $expr_{21}$ ... (id_k `=`)? $expr_{2k}$`)`], σ)

ELIF σ(**"checkpoint"**) $= id_t$ **AND** $expr_1 = id_t$`.save` **THEN**

([`if hvd.rank() == 0:` [$expr_1$ `(`$expr_{11}$ ... $expr_{1n}$ (id_1 `=`)? $expr_{21}$ ... (id_k `=`)? $expr_{2k}$`)`]], σ)

ELIF σ(**"model"**) $= id_t$ **AND** $expr_1 = id_t$`.fit` **THEN**

IF $id_i =$ `verbose` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**

IF $id_j =$ `callbacks` **WHEN** $1 \leq \mathrm{j} \leq \mathrm{k}$ **THEN**

([`callback = [hvd.callbacks.BroadcastGlobalVariablesCallback(root_rank=0)]`

if hvd.rank() == 0: callbacks.append($expr_{2j}$)
$expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... id_i = 1 if hvd.rank() == 0 else 0
... id_j = callbacks ... (id_k =)? $expr_{2k}$)], σ)
ELSE
([$expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ...
id_i = 1 if hvd.rank() == 0 else 0 ... (id_k =)? $expr_{2k}$
callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(root_rank=0)])], σ)
ELSE
IF id_j $=$ callbacks **WHEN** $1 \leq \mathrm{j} \leq \mathrm{k}$ **THEN**
([callback = [hvd.callbacks.BroadcastGlobalVariablesCallback(root_rank=0)]
if hvd.rank() == 0: callbacks.append($expr_{2j}$)
$expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... id_j = callbacks ... (id_k =)? $expr_{2k}$
verbose = 1 if hvd.rank() == 0 else 0)], σ)
ELSE
([$expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$
verbose = 1 if hvd.rank() == 0 else 0
callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(0)])], σ)
ELIF σ(**"model"**) $=$ id_t **AND** $expr_1$ $=$ id_t.compile **THEN**
IF id_i $=$ optimizer **AND** $expr_{2i}$ $=$ "adam" **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**
([id_z = tf.optimizers.Adam(learning_rate=0.001 * hvd.size())
id_z = hvd.DistributedOptimizer(id_z)
$expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... id_i = id_z ... (id_k =)? $expr_{2k}$)], σ)
ELIF $expr_{11}$ $=$ "adam" **THEN**
([id_z = tf.optimizers.Adam(learning_rate=0.001 * hvd.size())
id_z = hvd.DistributedOptimizer(id_z)
$expr_1$ (id_z ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$)], σ)
ELSE
[$trans_E$⟦ $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$) ⟧(σ)], σ)
ELIF $expr_1$ $=$ model.summary **THEN**
([if hvd.rank() == 0: $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$)], σ)
ELSE
[$trans_E$⟦ $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$) ⟧(σ)], σ)

A general form of expr statements except for the strict form

$trans_S⟦\ expr\ ⟧(\sigma) = ([trans_E⟦\ expr\ ⟧(\sigma)], \sigma)$

$trans_S⟦$ pass $⟧(\sigma) = ($[pass], $\sigma)$

$trans_S⟦$ break $⟧(\sigma) = ($[break], $\sigma)$

$trans_S⟦$ continue $⟧(\sigma) = ($[continue], $\sigma)$

$\boxed{trans_E \ : \ Expr \ \rightarrow \ \Sigma \ \rightarrow \ Expr}$

$$trans_E[\![\ expr_1 \quad boolop \ \ expr_2\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma) \quad boolop \ \ trans_E[\![\ expr_2\ ]\!](\sigma)$$

$$trans_E[\![\ expr_1 \quad \texttt{:=} \quad expr_2\ ]\!](\sigma) = expr_1 \quad \texttt{:=} \quad trans_E[\![\ expr_2\ ]\!](\sigma)$$

$$trans_E[\![\ expr_1 \quad binop \ \ expr_2\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma) \quad binop \ \ trans_E[\![\ expr_2\ ]\!](\sigma)$$

$$trans_E[\![\ unop \ \ expr\ ]\!](\sigma) = unop \ \ trans_E[\![\ expr\ ]\!](\sigma)$$

$$trans_E[\![\ \texttt{lambda} \ \ args \ \ \texttt{:} \ \ expr\ ]\!](\sigma) = \texttt{lambda} \ \ args \ \ \texttt{:} \ \ trans_E[\![\ expr\ ]\!](\sigma)$$

$$\begin{aligned}&trans_E[\![\ expr_1 \quad \texttt{if} \ \ expr_2 \quad \texttt{else} \ \ expr_3\ ]\!](\sigma) = \\ &\quad trans_E[\![\ expr_1\ ]\!](\sigma) \quad \texttt{if} \ \ trans_E[\![\ expr_2\ ]\!](\sigma) \quad \texttt{else} \ \ trans_E[\![\ expr_3\ ]\!](\sigma)\end{aligned}$$

$$\begin{aligned}&trans_E[\![\ expr_1 \quad \texttt{if} \ \ expr_2 \quad \texttt{else} \ \ expr_3\ ]\!](\sigma) = \\ &\quad trans_E[\![\ expr_1\ ]\!](\sigma) \quad \texttt{if} \ \ trans_E[\![\ expr_2\ ]\!](\sigma) \quad \texttt{else} \ \ trans_E[\![\ expr_3\ ]\!](\sigma)\end{aligned}$$

$$trans_E[\![\ \{(expr_1 \ : \ expr_2)^* \ \ (**expr_3)^*\}\ ]\!](\sigma) = \{(expr_1 \ : \ trans_E[\![expr_2]\!](\sigma))^* \ \ (**expr_3)^*\}$$

$$trans_E[\![\ \{expr^*\}\ ]\!](\sigma) = \{(trans_E[\![expr]\!](\sigma))^*\}$$

$$trans_E[\![\ \texttt{[}expr^*\texttt{]}\ ]\!](\sigma) = \texttt{[}(trans_E[\![expr]\!](\sigma))^*\texttt{]}$$

$$trans_E[\![\ \texttt{(}expr^*\texttt{)}\ ]\!](\sigma) = \texttt{(}(trans_E[\![expr]\!](\sigma))^*\texttt{)}$$

$$trans_E[\![\ \{expr_1\texttt{:}expr_2 \ \ comprehension^*\}\ ]\!](\sigma) = \{expr_1\texttt{:}trans_E[\![\ \ expr_2\ \ ]\!]\texttt{(}\sigma\texttt{)} \ \ (trans_O[\![comprehension]\!](\sigma))^*\}$$

$$trans_E[\![\ \{expr \quad comprehension^*\}\ ]\!](\sigma) = \{trans_E[\![\ \ expr\ \ ]\!]\texttt{(}\sigma\texttt{)} \ \ (trans_O[\![comprehension]\!](\sigma))^*\}$$

$$trans_E[\![\ \texttt{[}expr \quad comprehension^*\texttt{]}\ ]\!](\sigma) = \texttt{[}trans_E[\![\ \ expr\ \ ]\!]\texttt{(}\sigma\texttt{)} \ \ (trans_O[\![comprehension]\!](\sigma))^*\texttt{]}$$

$$trans_E[\![\ \texttt{(}expr \ \ comprehension^*\texttt{)}\ ]\!](\sigma) = \texttt{(}trans_E[\![\ expr\ ]\!](\sigma) \ (trans_O[\![comprehension]\!](\sigma))^*\texttt{)}$$

$$trans_E[\![\ \texttt{await} \ \ expr\ ]\!](\sigma) = \texttt{await} \ \ trans_E[\![\ expr\ ]\!](\sigma)$$

$$trans_E[\![\ \texttt{yield} \ \ expr?\ ]\!](\sigma) = \texttt{yield} \ \ (trans_E[\![\ expr\ ]\!](\sigma))?$$

$$trans_E[\![\ \texttt{yield from} \ \ expr\ ]\!](\sigma) = \texttt{yield from} \ \ trans_E[\![\ expr\ ]\!](\sigma)$$

$$trans_E[\![\ expr_1 \ (compop \ expr_2)^*\ ]\!](\sigma) = trans_E[\![\ expr_1\ ]\!](\sigma) \ (compop \ trans_E[\![expr_2]\!](\sigma))^*$$

$trans_E[\![$ $expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$) $]\!](\sigma) =$

IF σ(**"dataset"**) = id_t **AND** $expr_1$ = id_t`.take` **THEN**

IF id_i = `count` **WHEN** $1 \leq \text{i} \leq \text{k}$ **THEN**

$expr_1$ ($expr_{11}$ ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... id_i = $expr_{2i}$ // `hvd.size()` ... (id_k =)? $expr_{2k}$)

ELSE

$expr_1$ ($expr_{11}$ // `hvd.size()` ... $expr_{1n}$ (id_1 =)? $expr_{21}$ ... (id_k =)? $expr_{2k}$)

ELSE

$trans_E[\![$ $expr_1$ $]\!](\sigma)$ ($trans_E[\![$ $expr_{11}$ $]\!](\sigma)$ ... $trans_E[\![$ $expr_{1n}$ $]\!](\sigma)$

(id_1 =)? $trans_E[\![$ $expr_{21}$ $]\!](\sigma)$ ... (id_k =)? $trans_E[\![$ $expr_{2k}$ $]\!](\sigma)$)

$trans_E[\![$ {$expr_1$ (!i)? (:$expr_2$)?} $]\!](\sigma) =$ {$expr_1$ (!i)? (:$expr_2$)?}

$trans_E[\![$ $expr^*$ $]\!](\sigma) = expr^*$

$trans_E[\![$ $constant$ $]\!](\sigma) = constant$

$trans_E[\![$ $expr.id$ $]\!](\sigma) = (trans_E[\![$ $expr$ $]\!](\sigma)).id$

$trans_E[\![$ $expr_1$[$expr_2$] $]\!](\sigma) = trans_E[\![$ $expr_1$ $]\!](\sigma)$[$trans_E[\![$ $expr_2$ $]\!](\sigma)$]

$trans_E[\![$ *$expr$ $]\!](\sigma) =$ *$expr$

$trans_E[\![$ **$expr$ $]\!](\sigma) =$ **$expr$

$trans_E[\![$ id $]\!](\sigma) = id$

$trans_E[\![$ $expr_1$? (:$expr_2$)? (:$expr_3$)? $]\!](\sigma) = trans_E[\![$ $expr_1$ $]\!](\sigma)$? (:$trans_E[\![$ $expr_2$ $]\!](\sigma)$)? (:$trans_E[\![$ $expr_3$ $]\!](\sigma)$)?

$\boxed{trans_O \;:\; Comprehension \;\rightarrow\; \Sigma \;\rightarrow\; Comprehension}$

$trans_O[\![$ `for` $expr_1$ `in` $expr_2$ (`if` $expr_3$)* $]\!](\sigma) =$ `for` $expr_1$ `in` $trans_E[\![$ $expr_2$ $]\!](\sigma)$ (`if` $trans_E[\![expr_3]\!](\sigma)$)*

$\boxed{trans_H \;:\; ExcHandler \;\rightarrow\; \Sigma \;\rightarrow\; ExcHandler}$

$trans_H[\![$ `except` $expr$? (`as` id)? : $stmt^*$ $]\!](\sigma) =$ `except` $expr$? (`as` id)? : $trans_{\overline{S}}[\![$ $stmt^*$ $]\!](\sigma)$._**1**

$\boxed{trans_{\overline{A}} \;:\; Alias \;\textbf{list} \;\rightarrow\; \Sigma \;\rightarrow\; \Sigma}$

$trans_{\overline{A}}[\![$ $alias_1$ ... $alias_{n-1}$ $alias_n$ $]\!](\sigma) = trans_A[\![$ $alias_n$ $]\!](trans_A[\![$ $alias_{n-1}$ $]\!](\ldots$ $trans_A[\![$ $alias_1$ $]\!](\sigma)))$

$\boxed{trans_A \;:\; Alias \;\rightarrow\; \texttt{CG} \;\rightarrow\; \texttt{CG}}$

$trans_A[\![\ id_1\ .id_2\ ...\ .id_n\ ]\!](\texttt{cg}) =$

$\textbf{LET}\ \ (\texttt{ns}, \texttt{vs})\ \ = \texttt{CG}\ \ \textbf{IN}$

$\quad \textbf{LET}\ \ node_1 = \{\ id_1, \cdot\ \}, node_i = \{\ id_i, node_{i-1}\ \}\ \textbf{WHEN}\ \ 2 \le i \le n\ \textbf{IN}$

$\quad \textbf{LET}\ \ var_k = id_1\ ...\ id_k\ \textbf{WHEN}\ \ 1 \le k \le n\ \ \textbf{IN}$

$(\texttt{ns}\textbf{@}[node_1, ..., node_n], \texttt{vs}\textbf{@}[var_1 \mapsto node_1, ..., var_n \mapsto node_n])$

$trans_A[\![\ id_1\ .id_2\ ...\ .id_n\ \texttt{as}\ \ id_{as}\ ]\!](\texttt{cg}) =$

$\textbf{LET}\ \ (\texttt{ns}, \texttt{vs})\ \ = \texttt{CG}\ \ \textbf{IN}$

$\quad \textbf{LET}\ \ node_1 = \{\ id_1, \cdot\ \}, node_i = \{\ id_i, node_{i-1}\ \}\ \textbf{WHEN}\ \ 2 \le i \le n\ \textbf{IN}$

$(\texttt{ns}\textbf{@}[node_1, ..., node_n], \texttt{vs}\textbf{@}[id_{as} \mapsto node_n])$

$\boxed{trans_{\overline{W}}\ \ \textbf{:}\ \ WithItem\ \ \textbf{list}\ \ \rightarrow\ \ \Sigma\ \rightarrow\ \ (WithItem\ \ \textbf{list} \times \Sigma)}$

$trans_{\overline{W}}[\![\ with\_item_1\ with\_item_2\ ...\ with\_item_n\ ]\!](\sigma) =$

$\quad \textbf{LET}\ \ with\_item_1', \sigma_1\ \ = \ trans_W[\![\ with\_item_1\ ]\!](\sigma)\ \ \ \textbf{IN}$

$\quad \textbf{LET}\ \ with\_item_2', \sigma_2\ \ = \ trans_W[\![\ with\_item_2\ ]\!](\sigma_1)\ \ \ \textbf{IN}$

inden . . .

$\quad \textbf{LET}\ \ with\_item_n', \sigma_n\ \ = \ trans_W[\![\ with\_item_n\ ]\!](\sigma_{n-1})\ \ \ \textbf{IN}$

$\quad (with\_item_1'\ \textbf{::}\ with\_item_2'\ \textbf{::}\ ...\ \textbf{::}\ [with\_item_n'], \sigma_n)$

$\boxed{trans_W\ \ \textbf{:}\ \ WithItem\ \ \rightarrow\ \ \Sigma\ \rightarrow\ \ (WithItem\ \ \times \Sigma)}$

$trans_W[\![\ expr\ ]\!](\sigma) = (trans_E[\![\ expr\ ]\!](\sigma), \sigma)$

$trans_W[\![\ expr_1\ \texttt{as}\ \ expr_2\ ]\!](\text{smodenv}) =$

$\quad \textbf{IF}\ \ \sigma(\textbf{“tensor\_flow”})\ \ = \ id_1\ \ \ \textbf{AND}\ \ expr_1\ \ = \ id_1\texttt{.GradientTape()}\ \ \ \textbf{AND}\ \ expr_2\ \ = \ id_2\ \ \ \textbf{THEN}$

$\qquad (expr_1\ \texttt{as}\ \ expr_2, \sigma[\textbf{“gradient\_tape”} \mapsto id_2])$

$\quad \textbf{ELSE}\ \ (trans_E[\![\ expr_1\ ]\!](\sigma)\ \texttt{as}\ \ expr_2, \sigma)$

$\boxed{trans_C\ \ \textbf{:}\ \ MatchCase\ \ \rightarrow\ \ \Sigma\ \rightarrow\ \ MatchCase}$

$trans_C[\![\ \texttt{case}\ \ pattern\ \ (\texttt{if}\ \ expr)?\ :\ \ stmt^*\ ]\!](\sigma) =$

$\quad \texttt{case}\ \ trans_P[\![\ pattern\ ]\!](\sigma)\ \ \ (\texttt{if}\ \ trans_E[\![\ expr\ ]\!](\sigma))?\ :\ \ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)\textbf{.\_1}$

$\boxed{trans_P\ \ \textbf{:}\ \ Pattern\ \ \rightarrow\ \ \Sigma\ \rightarrow\ \ Pattern}$

$trans_P[\![\ expr\ ]\!](\sigma) = trans_E[\![\ expr\ ]\!](\sigma)$

$trans_P[\![\ constant\ ]\!](\sigma) = constant$

$trans_P[\![\ \texttt{[}pattern^*\texttt{]}\ ]\!](\sigma) = \texttt{[}trans_P[\![pattern]\!](\sigma)^*\texttt{]}$

$trans_P[\![\ \texttt{*}(id)?\ ]\!](\sigma) = \texttt{*}(id)?$

$trans_P[\![\ \{(expr\ :\ pattern)^*\ \ id?\}\ ]\!](\sigma) = \{(expr\ :\ trans_P[\![pattern]\!](\sigma))^*\ \ id?\}$

$$trans_P[\![\ expr\ \ (pattern_1^*\ (id = pattern_2)^*)\ ]\!](\sigma) = expr\ \ (trans_P[\![pattern_1]\!](\sigma)^*\ (id = trans_P[\![pattern_2]\!](\sigma))^*)$$

$$trans_P[\![\ (pattern\ \ \texttt{as})?\ id\ ]\!](\sigma) = (trans_P[\![\ pattern\ ]\!](\sigma)\ \texttt{as})?\ id$$

$$trans_P[\![\ pattern_1\ \texttt{|}\ pattern_2\ ]\!](\sigma) = trans_P[\![\ pattern_1\ ]\!](\sigma)\ \texttt{|}\ trans_P[\![\ pattern_2\ ]\!](\sigma)$$

$$trans_P[\![\ \_\ ]\!](\sigma) = \_$$

3 Identifying trainig loop

3.1 Restrictions

1. Training loop must be defined in only one file.
2. Training loop type is either `distributed gradient tape` or `distributed optimizer`.
3. Each model must have only one type of training loop.
4. Function must not be assigned to the variables or passed as an argument.
5. Training loop must not be defined conditionally.

3.2 Rules

3.2.1 Summary

$$\begin{array}{lcll}
Summary & ::= & \texttt{ModuleSummary}\ \ id\ \ \sigma_\top\ \ tl & (\textsc{ModuleSummary}) \\
 & & \texttt{FuncSummary}\ \ tl & (\textsc{FuncSummary}) \\
 & & \texttt{ClassSummary}\ \ arg & (\textsc{ClassSummary}) \\
 & & \texttt{ValueSummary}\ \ s & (\textsc{ValueSummary}) \\
arg & ::= & Model \mid \bot & (\textsc{Arg}) \\
tl & ::= & GradTape \mid Optimizer \mid \bot & (\textsc{TrainingLoop}) \\
\sigma_\top & ::= & \sigma\ \cup \top &
\end{array}$$

$\sigma\ \in \Sigma\ = Id \overset{\text{fin}}{\to} Summary$ Environment storing mappings from ids to summaries

3.2.2 Training Loop

$$\boxed{summary_M\ \ \mathbf{:}\ \ Module\ \to\ \Sigma\ \to\ ModuleSummary}$$

$$\begin{array}{l}
summary_M[\![\ id\ \ stmt^*\ \ \ type_ignore\ ]\!] = \\
\quad \textbf{LET}\ \ \sigma, tl\ \ =\ \ summary_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)\ \ \ \textbf{IN} \\
\quad \texttt{ModuleSummary}\ \ id\ \ \sigma\ \ tl
\end{array}$$

$$\boxed{summary_{\overline{S}}\ \ \mathbf{:}\ \ Stmt\ \ \textbf{list}\ \ \to\ \ \Sigma\ \ \to\ \ (\Sigma\ \ \times\ \ tl)}$$

$$
\begin{aligned}
summary_{\overline{S}}[\![\ stmt_1 \quad stmt_2 \ ...\ stmt_n\ ]\!](\sigma) \quad = \quad & \textbf{LET}\ \ \sigma_1, tl_1 \ \ = \ summary_S[\![\ stmt_1\ ]\!](\sigma) \quad \textbf{IN} \\
& \textbf{LET}\ \ \sigma_2, tl_2 \ \ = \ summary_{\overline{S}}[\![\ stmt_2\ ]\!](\sigma_1) \quad \textbf{IN} \\
& \ldots \\
& \textbf{LET}\ \ \sigma_n, tl_n \ \ = \ summary_{\overline{S}}[\![\ stmt_n\ ]\!](\sigma_{n-1}) \quad \textbf{IN} \\
& (\sigma_n, tl_1 \sqcup tl_2 \sqcup \ \ \ldots\ tl_n)
\end{aligned}
$$

$$\boxed{summary_S \ \ \textbf{:} \ \ Stmt \ \rightarrow \ \Sigma \ \rightarrow \ (\Sigma \ \times \ tl)}$$

$summary_S[\![\ (@expr_1)^*\ \ \texttt{def}\ \ id\ \ (args)\ \ (\texttt{->}\ expr_2)?\ \ \texttt{:}\ \ (\texttt{\#type:}s)?\ \ stmt^*\ ]\!](\sigma) =$
$\quad \textbf{LET}\ \ \sigma', tl\ = summary_{\overline{S}}[\![\ stmt^*\ ]\!]\quad \textbf{IN}$
$\quad (\sigma[id \mapsto \texttt{FuncSummary}\ \ tl], \perp)$

$summary_S[\![\ (@expr_1)^*\ \ \texttt{async def}\ \ id\ \ (args)\ \ (\texttt{->}\ expr_2)?\ \ \texttt{:}\ \ (\texttt{\#type:}s)?\ \ stmt^*\ ]\!](\sigma) =$
$\quad \textbf{LET}\ \ \sigma', tl\ = summary_{\overline{S}}[\![\ stmt^*\ ]\!]\quad \textbf{IN}$
$\quad (\sigma[id \mapsto \texttt{FuncSummary}\ \ tl], \perp)$

$summary_S[\![\ (@expr)^*\ \ \texttt{class}\ \ id\ \ (expr_{11}\ ...\ expr_{1n}\ \ \ (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k})\ \ \ \texttt{:}\ \ stmt^*\ ]\!](\sigma) =$
$\quad \textbf{IF}\ \ expr_{1i}\ \ =\ \ id_c\texttt{.keras.Model}\ \ \ \textbf{WHEN}\ \ 1 \le \mathrm{i} \le \mathrm{n}\ \ \ \textbf{AND}$
$\qquad \sigma(id_c)\ \ =\ \texttt{ModuleSummary}\ \ \mathrm{Id}(\text{"tensorflow"})\ \ \ [\,]\ \ \ \perp\ \textbf{THEN}\ \ (\sigma[id \mapsto \texttt{ClassSummary}\ \ Model], \perp)$
$\quad \textbf{ELIF}\ \ expr_{2i}\ \ =\ \ id_c\texttt{.keras.Model}\ \ \ \textbf{WHEN}\ \ 1 \le \mathrm{i} \le \mathrm{k}\ \ \ \textbf{AND}$
$\qquad \sigma(id_c)\ \ =\ \texttt{ModuleSummary}\ \ \mathrm{Id}(\text{"tensorflow"})\ \ \ [\,]\ \ \ \perp\ \textbf{THEN}\ \ (\sigma[id \mapsto \texttt{ClassSummary}\ \ Model], \perp)$
$\quad \textbf{ELSE}\ \ (\sigma[id \mapsto \texttt{ClassSummary}\ \ \perp], \perp)$

A strict form of assignment statements

$summary_S[\![\ id_r\ \texttt{=}\ id_c\ \texttt{()}\ (\texttt{\#type:}s)?\ ]\!](\sigma) =$
$\quad \textbf{IF}\ \ \sigma(id_c)\ \ =\ \texttt{ClassSummary}\ \ Model\ \ \textbf{THEN}\ \ (\sigma[id_r \mapsto \texttt{ValueSummary}\ \ \textbf{"model\_instance"}], \perp)$
$\quad \textbf{ELSE}\ \ (\sigma, \perp)$

$summary_S[\![\ (\texttt{\#type:}s)?\ \ \texttt{for}\ \ expr_1\ \ \texttt{in}\ \ expr_2\ \ \texttt{:}\ \ stmt_1^*\ \ \ (\texttt{else}\ \ \texttt{:}\ \ stmt_2^*)?\ ]\!](\sigma) =$
$\quad \textbf{LET}\ \ \_, tl\ \ =\ \ summary_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)\quad \textbf{IN}$
$\quad (\sigma, tl)$

$summary_S[\![\ (\texttt{\#type:}s)?\ \ \texttt{async for}\ \ expr_1\ \ \texttt{in}\ \ expr_2\ \ \texttt{:}\ \ stmt_1^*\ \ \ (\texttt{else}\ \ \texttt{:}\ \ stmt_2^*)?\ ]\!](\sigma) =$
$\quad \textbf{LET}\ \ \_, tl\ \ =\ \ summary_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)\quad \textbf{IN}$
$\quad (\sigma, tl)$

$summary_S[\![\ \texttt{while}\ \ (expr)\ \ \texttt{:}\ \ stmt_1^*\ \ \ (\texttt{else}\ \ \texttt{:}\ \ stmt_2^*)?\ ]\!](\sigma) =$
$\quad \textbf{LET}\ \ \_, tl\ \ =\ \ summary_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)\quad \textbf{IN}$
$\quad (\sigma, tl)$

$summary_S[\![\ (\texttt{\#type:}s)?\ \ \texttt{with}\ \ with\_item^*\ \ \texttt{:}\ \ stmt^*\ ]\!](\sigma) =$
$\quad \textbf{LET}\ \ \_, tl\ \ =\ \ summary_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)\quad \textbf{IN}$
$\quad (\sigma, summary_{\overline{W}}[\![\ with\_item^*\ ]\!](\sigma) \sqcup\ \ tl)$

$summary_S[\![\ (\texttt{\#type:}s)?\ \ \texttt{async with}\ \ with\_item^*\ \ \texttt{:}\ \ stmt^*\ ]\!](\sigma) =$
$\quad \textbf{LET}\ \ \_, tl\ \ =\ \ summary_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)\quad \textbf{IN}$
$\quad (\sigma, summary_{\overline{W}}[\![\ with\_item^*\ ]\!](\sigma) \sqcup\ \ tl)$

$$summary_S[\![\, \texttt{try} \;\; \texttt{:} \;\; stmt_1^* \;\; exc\_handler^* \;\; (\texttt{else} \;\; \texttt{:} \;\; stmt_2^*)? \;\; (\texttt{finally} \;\; \texttt{:} \;\; stmt_3^*)? \,]\!](\sigma) = \\ (\sigma,\, summary_{\overline{S}}[\![\, stmt_1^* \,]\!](\sigma))$$

$$summary_S[\![\, \texttt{import} \;\; alias^* \,]\!](\sigma) = (summary_{\overline{A}}[\![\, alias^* \,]\!](\sigma),\, \perp)$$

$$\begin{array}{l} summary_S[\![\, \texttt{from} \;\; 0 \;\; id^* \;\; \texttt{import} \;\; alias^* \,]\!](\sigma) = \\ \quad \textbf{LET} \;\; \sigma' \;\; = \;\; summary_{\overline{A}}[\![\, alias^* \,]\!](\sigma) \;\; \textbf{IN} \\ \quad \textbf{LET} \;\; [id_2 \mapsto \texttt{ModuleSummary} \;\; id_2 \; \sigma_2 \perp, \;\; id_2 \mapsto \texttt{ModuleSummary} \; id_2 \; \sigma_2 \perp, \\ \quad\quad \textbf{...} \; id_n \mapsto \texttt{ModuleSummary} \;\; id_n \; \sigma_n \perp] \;\; = \;\; \sigma' \;\; \backslash \;\; \sigma \;\; \textbf{IN} \\ \quad \sigma \;\; ++ \;\; [id_2 \mapsto \texttt{ModuleSummary} \;\; (id^* + id_2) \; \sigma_2 \perp, \;\; id_2 \mapsto \texttt{ModuleSummary} \;\; (id^* + id_2) \; \sigma_2 \perp, \\ \quad\quad \textbf{...} \; id_n \mapsto \texttt{ModuleSummary} \;\; (id^* + id_n) \; \sigma_n \perp] \end{array}$$

$$summary_S[\![\, expr \,]\!](\sigma) = (\sigma,\, summary_E[\![\, expr \,]\!](\sigma))$$

$$summary_S[\![\, stmt \,]\!](\sigma) = (\sigma,\, \perp)$$

$$\boxed{summary_{\overline{A}} \;\; \textbf{:} \;\; Alias \;\; \textbf{list} \;\; \rightarrow \;\; \Sigma \;\; \rightarrow \;\; \Sigma}$$

$$\begin{array}{l} summary_{\overline{A}}[\![\, alias_1 \; ... \; alias_{n-1} \; alias_n \,]\!](\sigma) = \\ \quad summary_A[\![\, alias_n \,]\!](summary_A[\![\, alias_{n-1} \,]\!](... \;\; summary_A[\![\, alias_1 \,]\!](\sigma) \; ...)) \end{array}$$

$$\boxed{trans_A \;\; \textbf{:} \;\; Alias \;\; \rightarrow \;\; \Sigma \;\; \rightarrow \;\; \Sigma}$$

$$\begin{array}{l} trans_A[\![\, id_1 \; id_2 \; ... \; id_n \,]\!](\sigma) = \\ \quad \textbf{LET} \;\; \sigma_n = [id_n \mapsto \texttt{ModuleSummary} \;\; id_n \;\;\; \top \;\; \perp] \\ \quad \textbf{LET} \;\; \sigma_{n-1} = [id_{n-1} \mapsto \texttt{ModuleSummary} \;\; id_{n-1} \;\;\; \sigma_n \;\;\; \perp] \\ \quad \textbf{...} \\ \quad \textbf{LET} \;\; \sigma_2 = [id_2 \mapsto \texttt{ModuleSummary} \;\; id_2 \;\;\; \sigma_3 \;\;\; \perp] \\ \quad \textbf{LET} \;\; \sigma_1 = [id_1 \mapsto \texttt{ModuleSummary} \;\; id_1 \;\;\; \sigma_2 \;\;\; \perp] \\ \quad \sigma \;\; ++ \;\; \sigma_1 \end{array}$$

$$\boxed{summary_{\overline{W}} \;\; \textbf{:} \;\; WithItem \;\; \textbf{list} \;\; \rightarrow \;\; \Sigma \;\; \rightarrow \;\; tl}$$

$$\begin{array}{l} summary_{\overline{W}}[\![\, with\_item_1 \; with\_item_2 \; ... \; with\_item_n \,]\!](\sigma) = \\ \quad summary_W[\![\, with\_item_1 \,]\!](\sigma) \sqcup \;\; summary_W[\![\, with\_item_2 \,]\!](\sigma) \sqcup \;\; \textbf{...} \;\;\; summary_W[\![\, with\_item_n \,]\!](\sigma) \end{array}$$

$$\boxed{summary_W \;\; \textbf{:} \;\; WithItem \;\; \rightarrow \;\; \Sigma \;\; \rightarrow \;\; tl}$$

$$summary_W[\![\, expr_1 \; \texttt{as} \;\; expr_2? \,]\!](\sigma) = summary_E[\![\, expr_1 \,]\!](\sigma)$$

$$\boxed{summary_E \;\; \textbf{:} \;\; Expr \;\; \rightarrow \;\; \Sigma \;\; \rightarrow \;\; tl}$$

A strict form of call expression

$$
\begin{array}{l}
summary_E[\![\ expr_1\ \texttt{(}expr_{11}\ ...\ expr_{1n}\quad (id_1\ \texttt{=}\ )?\ expr_{21}\ ...\ (id_k\ \texttt{=}\ )?\ expr_{2k}\texttt{)}\ ]\!](\sigma) = \\
\quad \textbf{IF}\ \ \sigma(id_r)\ \ =\ \texttt{ValueSummary}\ \ \textbf{"model\_instance"}\ \ \ \textbf{AND} \\
\quad\quad expr_1\ \ =\ id_r\texttt{.fit}\ \ \ \textbf{THEN}\ \ Optimizer \\
\quad \textbf{ELIF}\ \ \sigma(id_r)\ \ =\ \texttt{ModuleSummary}\ \ \mathrm{Id}(\text{"tensorflow"})\ \ \ \sigma'\ \ \ tl\ \ \textbf{AND} \\
\quad\quad expr_1\ \ =\ id_r\texttt{.GradientTape}\ \ \ \textbf{THEN}\ \ GradTape \\
\quad \textbf{ELIF}\ \ \sigma(id_r)\ \ =\ \texttt{FuncSummary}\ \ tl\ \ \textbf{AND} \\
\quad\quad expr_1\ \ =\ \texttt{idr}\ \ \ \textbf{THEN}\ \ tl \\
\quad \textbf{ELSE}\ \ \bot
\end{array}
$$

$$summary_E[\![\ expr\ ]\!](\sigma) = \bot$$