Code Transformation for Distributed Python ML Code

1 Python Abstract Syntax

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

$$
\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{(!}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}?\ 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}
$$

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

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

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

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

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

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

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

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

._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

2.2.2 Transformation Rules

$$\boxed{trans_M \;\; \mathbf{:} \;\; Module \;\rightarrow\; Module}$$

$$trans_M[\![\; stmt^* \quad type\_ignore \;]\!] \;=\; trans_{\overline{S}}[\![\; stmt^* \;]\!](\sigma)\mathbf{.\_1}\;\; type\_ignore$$

$$\boxed{trans_{\overline{S}} \;\; \mathbf{:} \;\; Stmt\ \textbf{list} \;\rightarrow\; \Sigma \;\rightarrow\; (Stmt\ \textbf{list} \;\times\; \Sigma)}$$

$$\begin{array}{lll} trans_{\overline{S}}[\![\; stmt_1 \quad stmt_2 \;...\; stmt_n \;]\!](\sigma) & = & \textbf{LET}\;\; stmt_1^{*\prime}, \sigma_1 \;=\; trans_S[\![\; stmt_1 \;]\!](\sigma) \;\;\textbf{IN} \\ & & \textbf{LET}\;\; stmt_2^{*\prime}, \sigma_2 \;=\; trans_S[\![\; stmt_2 \;]\!](\sigma_1) \;\;\textbf{IN} \\ & & \ldots \\ & & \textbf{LET}\;\; stmt_n^{*\prime}, \sigma_n \;=\; trans_S[\![\; stmt_n \;]\!](\sigma_{n-1}) \;\;\textbf{IN} \\ & & (stmt_1^{*\prime}\;\textbf{@}\;stmt_2^{*\prime}\;\textbf{@}\;...\;\textbf{@}\;stmt_n^{*\prime}, \sigma_n) \end{array}$$

$$\boxed{trans_S \;\; \mathbf{:} \;\; Stmt \;\rightarrow\; \Sigma \;\rightarrow\; (Stmt\ \textbf{list} \;\times\; \Sigma)}$$

$trans_S[\![\; (@expr_1)^* \;\; \texttt{def} \;\; id \;\; \texttt{(}args\texttt{)} \;\; (\texttt{->}\; expr_2)? \;\; \texttt{:} \;\; (\texttt{\#type:}s)? \;\; stmt^* \;]\!](\sigma) =$
$\quad ([(@expr_1)^* \;\; \texttt{def} \;\; id \;\; \texttt{(}args\texttt{)} \;\; (\texttt{->}\; expr_2)? \;\; \texttt{:} \;\; (\texttt{\#type:}s)? \;\; trans_{\overline{S}}[\![\; stmt^* \;]\!](\sigma)\mathbf{.\_1}], \sigma)$

$trans_S[\![\; (@expr_1)^* \;\; \texttt{async def} \;\; id \;\; \texttt{(}args\texttt{)} \;\; (\texttt{->}\; expr_2)? \;\; \texttt{:} \;\; (\texttt{\#type:}s)? \;\; stmt^* \;]\!](\sigma) =$
$\quad ([(@expr_1)^* \;\; \texttt{async def} \;\; id \;\; \texttt{(}args\texttt{)} \;\; (\texttt{->}\; expr_2)? \;\; \texttt{:} \;\; (\texttt{\#type:}s)? \;\; trans_{\overline{S}}[\![\; stmt^* \;]\!](\sigma)\mathbf{.\_1}], \sigma)$

$trans_S[\![\; (@expr_1)^* \;\; \texttt{class} \;\; id \;\; \texttt{(}expr_2^*\; keyword^*\texttt{)} \;\; \texttt{:} \;\; stmt^* \;]\!](\sigma) =$
$\quad ([(@expr_1)^* \;\; \texttt{class} \;\; id \;\; \texttt{(}expr_2^*\; keyword^*\texttt{)} \;\; \texttt{:} \;\; trans_{\overline{S}}[\![\; stmt^* \;]\!](\sigma)\mathbf{.\_1}], \sigma) =$

$trans_S[\![\; \texttt{return} \;\; expr? \;]\!](\sigma) = ([\texttt{return} \;\; (trans_E[\![\; expr \;]\!](\sigma))?], \sigma)$

$trans_S[\![\; \texttt{delete} \;\; expr^* \;]\!](\sigma) = ([\texttt{delete} \;\; expr^*], \sigma)$

A strict form of assignment statements

$trans_S[\![\; id_r \texttt{ = } expr_1 \;\texttt{(}expr_{11} \;...\; expr_{1n} \;\; (id_1 \texttt{ = })?\; expr_{21} \;...\; (id_k \texttt{ = })?\; expr_{2k}\texttt{)}\; (\texttt{\#type:}s)? \;]\!](\sigma) =$
$\quad$ **IF** $\sigma(\textbf{“tensor\_flow”}) \;=\; id_t$ **AND** $expr_1 \;=\; id_t\texttt{.data.Dataset.}expr_3$ **THEN**
$\quad\quad ([id_r \texttt{ = } expr_1 \;\texttt{(}expr_{11} \;...\; expr_{1n} \;\; (id_1 \texttt{ = })?\; expr_{21} \;...\; (id_k \texttt{ = })?\; expr_{2k}\texttt{)}\; (\texttt{\#type:}s)?],$
$\quad\quad\quad \sigma[\textbf{“dataset”} \mapsto id_r])$
$\quad$ **ELIF** $\sigma(\textbf{“tensor\_flow”}) \;=\; id_t$ **AND** $expr_1 \;=\; id_t\texttt{.train.Checkpoint}$ **THEN**
$\quad\quad ([id_r \texttt{ = } expr_1 \;\texttt{(}expr_{11} \;...\; expr_{1n} \;\; (id_1 \texttt{ = })?\; expr_{21} \;...\; (id_k \texttt{ = })?\; expr_{2k}\texttt{)}\; (\texttt{\#type:}s)?],$
$\quad\quad\quad \sigma[\textbf{“checkpoint”} \mapsto id_r])$
$\quad$ **ELIF** $\sigma(\textbf{“tensor\_flow”}) \;=\; id_t$ **AND** $expr_1 \;=\; id_t\texttt{.optimizers.Adam}$ **THEN**
$\quad\quad$ **IF** $id_i \;=\; \texttt{learning\_rate}$ **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**
$\quad\quad\quad ([id_r \texttt{ = } expr_1 \;\texttt{(}expr_{11} \;...\; expr_{1n} \;\; (id_1 \texttt{ = })?\; expr_{21} \;...\; id_i \texttt{ = } expr_{2i} \texttt{ * hvd.size()}$

... (id_k `=`)? $expr_{2k}$`)` (`#type:`s)?], 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)?],

σ[**“optimizer”**$\mapsto$ id_r])

ELIF σ(**“optimizers”**) $=$ id_t **AND** $expr_1$ $=$ id_t`.Adam` **THEN**

IF id_i $=$ `learning_rate` **WHEN** $1 \leq \text{i} \leq \text{k}$ **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)?], 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)?],

σ[**“optimizer”**$\mapsto$ id_r])

ELIF σ(**“optimizer”**) $=$ id_t **AND** $expr_1$ $=$ id_t`.apply_gradients` **THEN**

IF id_i $=$ `grads_and_vars` **WHEN** $1 \leq \text{i} \leq \text{k}$ **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`]], σ)

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** $1 \leq \text{i} \leq \text{k}$ **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 `=`)? $expr_{2k}$`)` ⟧(σ) (`#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 $([], \sigma)$

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)?$], \sigma)$

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

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

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)?$], \sigma[$**"dataset"**$\mapsto id_1])$

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

$trans_S[\![$ (#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)?$], \sigma)$

$trans_S[\![$ (#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)?$], \sigma)$

$trans_S[\![$ while ($expr$) : $stmt_1^*$ (else : $stmt_2^*$)? $]\!](\sigma) =$

$([$while ($trans_E[\![\ expr\ ]\!](\sigma)$) : $trans_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)$._1 (else : $trans_{\overline{S}}[\![\ stmt_2^*\ ]\!](\sigma)$._1)?$], \sigma)$

$trans_S[\![$ if ($expr$) : $stmt^*$ (else : $stmt^*$)? $]\!](\sigma) =$

$([$if ($trans_E[\![\ expr\ ]\!](\sigma)$) : $trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)$._1 (else : $trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)$._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 $with\_item^{*\prime}$: $stmt^{*\prime}], \sigma_2)$

$$
\begin{array}{l}
trans_S[\![\ \texttt{match}\ \ expr\ \ \texttt{:}\ \ match\_case^*\ ]\!](\sigma) = \\
\quad ([\texttt{match}\ \ trans_E[\![\ expr\ ]\!](\sigma)\ \ \texttt{:}\ \ (trans_C[\![match\_case]\!](\sigma))^*],\ \sigma)
\end{array}
$$

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

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

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

$$
\begin{array}{l}
trans_S[\![\ \texttt{import}\ \ alias^*\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ \sigma_1\ = trans_{\overline{A}}[\![\ alias^*\ ]\!](\sigma)\ \textbf{IN} \\
\quad \textbf{IF}\ \ \sigma_1\ \setminus\ \sigma\ =\ [\textbf{``tensor\_flow''} \mapsto id]\ \ \textbf{THEN} \\
\qquad ([\texttt{import}\ \ alias^*, \\
\qquad \texttt{import horovod.tensorflow as hvd}, \\
\qquad \texttt{hvd\_broadcast\_done = False}, \\
\qquad \texttt{hvd.init()}, \\
\qquad \texttt{gpus = } id\texttt{.config.experimental.list\_physical\_devices('GPU')}, \\
\qquad \texttt{for gpu in gpus: } id\texttt{.config.experimental.set\_memory\_growth(gpu, True)}, \\
\qquad \texttt{if gpus: } id\texttt{.config.experimental.set\_visible\_devices(gpus[hvd.local\_rank()], 'GPU')}],\ \sigma_1) \\
\quad \textbf{ELSE}\ ([\texttt{import}\ \ alias^*],\ \sigma_1)
\end{array}
$$

$$
\begin{array}{l}
trans_S[\![\ \texttt{import}\ \ alias^*\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ \sigma_1\ = trans_{\overline{A}}[\![\ alias^*\ ]\!](\sigma)\ \textbf{IN} \\
\quad \textbf{IF}\ \ \sigma_1\ \setminus\ \sigma\ =\ [\textbf{``tensor\_flow''} \mapsto id]\ \ \textbf{THEN} \\
\qquad ([\texttt{import}\ \ alias^*, \\
\qquad \texttt{import horovod.tensorflow as hvd}, \\
\qquad \texttt{hvd.init()}, \\
\qquad \texttt{gpus = } id\texttt{.config.experimental.list\_physical\_devices('GPU')}, \\
\qquad \texttt{for gpu in gpus: } id\texttt{.config.experimental.set\_memory\_growth(gpu, True)}, \\
\qquad \texttt{if gpus: } id\texttt{.config.experimental.set\_visible\_devices(gpus[hvd.local\_rank()], 'GPU')}],\ \sigma_1) \\
\quad \textbf{ELSE}\ ([\texttt{import}\ \ alias^*],\ \sigma_1)
\end{array}
$$

$$
\begin{array}{l}
trans_S[\![\ \texttt{from}\ \ i\ \ id?\ \texttt{import}\ \ alias^*\ ]\!](\sigma) = \\
\quad \textbf{LET}\ \ \sigma_1\ = trans_{\overline{A}}[\![\ alias^*\ ]\!](\sigma)\ \textbf{IN} \\
\quad \textbf{IF}\ \ id\ =\ \texttt{tensorflow}\ \ \textbf{AND}\ \ \sigma_1\ \setminus\ \sigma\ =\ [\textbf{``keras''} \mapsto id]\ \ \textbf{THEN} \\
\qquad ([\texttt{from}\ \ i\ \ id?\ \texttt{import}\ \ alias^*, \\
\qquad \texttt{import horovod.tensorflow.keras as hvd}, \\
\qquad \texttt{hvd.init()}, \\
\qquad \texttt{gpus = } id\texttt{.config.experimental.list\_physical\_devices('GPU')}, \\
\qquad \texttt{for gpu in gpus: } id\texttt{.config.experimental.set\_memory\_growth(gpu, True)}, \\
\qquad \texttt{if gpus: } id\texttt{.config.experimental.set\_visible\_devices(gpus[hvd.local\_rank()], 'GPU')}],\ \sigma_1) \\
\quad \textbf{ELSE}\ \ ([\texttt{from}\ \ i\ \ id?\ \texttt{import}\ \ alias^*],\ \sigma_1)
\end{array}
$$

$trans_S⟦$ `global` id^* $⟧(\sigma) = ([$`global` $id^*], \sigma)$

$trans_S⟦$ `nonlocal` id^* $⟧(\sigma) = ([$`nonlocal` $id^*], \sigma)$

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 =)? $expr_{2k}$) $]\!](\sigma)$], 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 =)? $expr_{2k}$) $]\!](\sigma)$], σ)

A general form of expr statements except for the strict form

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

$trans_S[\![\ \texttt{pass}\ ]\!](\sigma) = ([\texttt{pass}], \sigma)$

$trans_S[\![\ \texttt{break}\ ]\!](\sigma) = ([\texttt{break}], \sigma)$

$trans_S[\![\ \texttt{continue}\ ]\!](\sigma) = ([\texttt{continue}], \sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IF $\sigma(\textbf{"dataset"})\ \ =\ id_t$ **AND** $expr_1\ \ =\ id_t\texttt{.take}$ **THEN**

  IF $id_i\ \ =\ \texttt{count}$ **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **THEN**

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

  ELSE

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

ELSE

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

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

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

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

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

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

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

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

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

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

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

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

$$trans_O[\![\ \texttt{for}\ \ expr_1\ \texttt{in}\ \ expr_2\ (\texttt{if}\ expr_3)^*\ ]\!](\sigma) = \texttt{for}\ \ expr_1\ \texttt{in}\ \ trans_E[\![\ expr_2\ ]\!](\sigma)\ (\texttt{if}\ trans_E[\![expr_3]\!](\sigma))^*$$

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

$$trans_H[\![\ \texttt{except}\ \ expr?\ (\texttt{as}\ \ id)?\ \texttt{:}\ \ stmt^*\ ]\!](\sigma) = \texttt{except}\ \ expr?\ (\texttt{as}\ \ id)?\ \texttt{:}\ \ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)\textbf{.\_1}$$

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

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

$$\boxed{trans_A \quad : \quad Alias \ \ \rightarrow \ \ \Sigma \ \ \rightarrow \ \ \Sigma}$$

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

LET id = `tensorflow` **THEN** $\sigma[$**“tensor_flow”**$\mapsto id]$
LET id = `keras` **THEN** $\sigma[$**“keras”**$\mapsto id]$
LET id = `os` **THEN** $\sigma[$**“os”**$\mapsto id]$
LET id = `optimizers` **THEN** $\sigma[$**“optimizers”**$\mapsto id]$
ELSE σ

$trans_A[\![\ id_1\ \texttt{as}\ \ id_2\ ]\!](\sigma) =$

LET id_1 = `tensorflow` **THEN** $\sigma[$**“tensor_flow”**$\mapsto id_2]$
LET id_1 = `tensorflow.keras` **THEN** $\sigma[$**“keras”**$\mapsto id_2]$
LET id_1 = `os` **THEN** $\sigma[$**“os”**$\mapsto id_2]$
LET id = `optimizers` **THEN** $\sigma[$**“optimizers”**$\mapsto id]$
ELSE σ

$trans_A[\![\ id_1\ .id_2\ (.id_3)^*\ (\texttt{as}\ \ id_2)?\ ]\!](\sigma) = \sigma$

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

$trans_{\overline{W}}[\![\ with\_item_1\ with\_item_2\ ...\ with\_item_n\ ]\!](\sigma) =$
$\quad\textbf{LET}\ \ with\_item_1', \sigma_1\ \ =\ trans_W[\![\ with\_item_1\ ]\!](\sigma)\ \ \textbf{IN}$
$\quad\textbf{LET}\ \ with\_item_2', \sigma_2\ \ =\ trans_W[\![\ with\_item_2\ ]\!](\sigma_1)\ \ \textbf{IN}$
$\quad\ldots$
$\quad\textbf{LET}\ \ with\_item_n', \sigma_n\ \ =\ trans_W[\![\ with\_item_n\ ]\!](\sigma_{n-1})\ \ \textbf{IN}$
$\quad(with\_item_1'\ \textbf{::}\ with\_item_2'\ \textbf{::}\ ...\ \textbf{::}\ [with\_item_n'], \sigma_n)$

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

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

$trans_W[\![\ expr_1\ \texttt{as}\ \ expr_2\ ]\!](\mathrm{smodenv}) =$
$\quad\textbf{IF}\ \ \sigma(\textbf{“tensor\_flow”})\ \ =\ id_1\ \ \textbf{AND}\ \ expr_1\ \ =\ id_1\texttt{.GradientTape()}\ \ \textbf{AND}\ \ expr_2\ \ =\ id_2\ \ \textbf{THEN}$
$\quad\quad(expr_1\ \texttt{as}\ \ expr_2, \sigma[\textbf{“gradient\_tape”}\mapsto id_2])$
$\quad\textbf{ELSE}\ \ (trans_E[\![\ expr_1\ ]\!](\sigma)\ \texttt{as}\ \ expr_2, \sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2.2 Training Loop

$$\boxed{summary_M\ \ :\ \ Module\ \rightarrow\ \Sigma\ \rightarrow\ ModuleSummary}$$

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

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

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

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

$summary_S⟦$ (@$expr_1$)* `def` id ($args$) (`->` $expr_2$)? : (`#type:`s)? $stmt^*$ $⟧(\sigma) =$

LET $\sigma', tl = summary_{\overline{S}}⟦ stmt^* ⟧$ **IN**

($\sigma[id \mapsto$ `FuncSummary` $tl], \perp)$

$summary_S⟦$ (@$expr_1$)* `async def` id ($args$) (`->` $expr_2$)? : (`#type:`s)? $stmt^*$ $⟧(\sigma) =$

LET $\sigma', tl = summary_{\overline{S}}⟦ stmt^* ⟧$ **IN**

($\sigma[id \mapsto$ `FuncSummary` $tl], \perp)$

$summary_S⟦$ (@$expr$)* `class` id ($expr_{11}$ ... $expr_{1n}$ (id_1 `=`)? $expr_{21}$ ... (id_k `=`)? $expr_{2k}$) : $stmt^*$ $⟧(\sigma) =$

IF $expr_{1i}$ = id_c`.keras.Model` **WHEN** $1 \leq$ i $\leq$ n **AND**

$\sigma(id_c)$ = `ModuleSummary` Id("tensorflow") [] $\perp$ **THEN** ($\sigma[id \mapsto$ `ClassSummary` $Model], \perp)$

ELIF $expr_{2i}$ = id_c`.keras.Model` **WHEN** $1 \leq$ i $\leq$ k **AND**

$\sigma(id_c)$ = `ModuleSummary` Id("tensorflow") [] $\perp$ **THEN** ($\sigma[id \mapsto$ `ClassSummary` $Model], \perp)$

ELSE ($\sigma[id \mapsto$ `ClassSummary` $\perp], \perp)$

A strict form of assignment statements

$summary_S⟦$ id_r `=` id_c `()` (`#type:`s)? $⟧(\sigma) =$

IF $\sigma(id_c)$ = `ClassSummary` $Model$ **THEN** ($\sigma[id_r \mapsto$ `ValueSummary` **"model_instance"**$], \perp)$

ELSE $(\sigma, \perp)$

$summary_S⟦$ (`#type:`s)? `for` $expr_1$ `in` $expr_2$: $stmt_1^*$ (`else` : $stmt_2^*$)? $⟧(\sigma) =$

LET _, tl = $summary_{\overline{S}}⟦ stmt_1^* ⟧(\sigma)$ **IN**

(σ, tl)

$summary_S⟦$ (`#type:`s)? `async for` $expr_1$ `in` $expr_2$: $stmt_1^*$ (`else` : $stmt_2^*$)? $⟧(\sigma) =$

LET _, tl = $summary_{\overline{S}}⟦ stmt_1^* ⟧(\sigma)$ **IN**

(σ, tl)

$summary_S⟦$ `while` ($expr$) : $stmt_1^*$ (`else` : $stmt_2^*$)? $⟧(\sigma) =$

LET _, tl = $summary_{\overline{S}}⟦ stmt_1^* ⟧(\sigma)$ **IN**

(σ, tl)

$summary_S⟦$ (`#type:`s)? `with` $with\_item^*$: $stmt^*$ $⟧(\sigma) =$

LET _, tl = $summary_{\overline{S}}⟦ stmt^* ⟧(\sigma)$ **IN**

$(\sigma, summary_{\overline{W}}⟦ with\_item^* ⟧(\sigma) \sqcup tl)$

$summary_S⟦$ (`#type:`s)? `async with` $with\_item^*$: $stmt^*$ $⟧(\sigma) =$

LET _, tl = $summary_{\overline{S}}⟦ stmt^* ⟧(\sigma)$ **IN**

$(\sigma, summary_{\overline{W}}⟦ with\_item^* ⟧(\sigma) \sqcup tl)$

$summary_S⟦$ `try` : $stmt_1^*$ $exc\_handler^*$ (`else` : $stmt_2^*$)? (`finally` : $stmt_3^*$)? $⟧(\sigma) =$

$(\sigma, summary_{\overline{S}}⟦ stmt_1^* ⟧(\sigma))$

$summary_S⟦$ `import` $alias^*$ $⟧(\sigma) = (summary_{\overline{A}}⟦ alias^* ⟧(\sigma), \perp)$

$$summary_S[\![\ \texttt{from}\ \ 0\ \ \ id^*\ \ \ \texttt{import}\ \ alias^*\ ]\!](\sigma) =$$
$$\textbf{LET}\ \ \sigma' \ \ = \ summary_{\overline{A}}[\![\ alias^*\ ]\!](\sigma)\ \ \ \textbf{IN}$$
$$\textbf{LET}\ \ [id_2 \mapsto \texttt{ModuleSummary}\ \ id_2\ \sigma_2\ \bot,\ \ \ id_2 \mapsto \texttt{ModuleSummary}\ id_2\ \sigma_2\ \bot,$$
$$\textbf{...}\ id_n \mapsto \texttt{ModuleSummary}\ \ id_n\ \sigma_n\ \bot] \ \ = \ \sigma' \ \ \ \setminus\ \ \sigma\ \ \textbf{IN}$$
$$\sigma\ \ {+}{+}\ \ \ [id_2 \mapsto \texttt{ModuleSummary}\ \ (id^*{+}id_2)\ \sigma_2\ \bot,\ \ \ id_2 \mapsto \texttt{ModuleSummary}\ \ (id^*{+}id_2)\ \sigma_2\ \bot,$$
$$\textbf{...}\ id_n \mapsto \texttt{ModuleSummary}\ \ (id^*{+}id_n)\ \sigma_n\ \bot]$$

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

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

$$\boxed{summary_{\overline{A}}\ \ \ \mathbf{:}\ \ \ Alias\ \ \textbf{list}\ \ \ \rightarrow\ \ \ \Sigma\ \ \rightarrow\ \ \ \Sigma}$$
$$summary_{\overline{A}}[\![\ alias_1\ ...\ alias_{n-1}\ alias_n\ ]\!](\sigma) =$$
$$summary_A[\![\ alias_n\ ]\!](summary_A[\![\ alias_{n-1}\ ]\!](...\ summary_A[\![\ alias_1\ ]\!](\sigma)\ ...))$$

$$\boxed{trans_A\ \ \ \mathbf{:}\ \ \ Alias\ \ \rightarrow\ \ \ \Sigma\ \ \rightarrow\ \ \ \Sigma}$$
$$trans_A[\![\ id_1\ id_2\ ...\ id_n\ ]\!](\sigma) =$$
$$\textbf{LET}\ \ \sigma_n = [id_n \mapsto \texttt{ModuleSummary}\ \ id_n\ \ \ \top\ \ \bot]$$
$$\textbf{LET}\ \ \sigma_{n-1} = [id_{n-1} \mapsto \texttt{ModuleSummary}\ \ id_{n-1}\ \ \ \sigma_n\ \ \ \bot]$$
$$\textbf{...}$$
$$\textbf{LET}\ \ \sigma_2 = [id_2 \mapsto \texttt{ModuleSummary}\ \ id_2\ \ \ \sigma_3\ \ \ \bot]$$
$$\textbf{LET}\ \ \sigma_1 = [id_1 \mapsto \texttt{ModuleSummary}\ \ id_1\ \ \ \sigma_2\ \ \ \bot]$$
$$\sigma\ \ {+}{+}\ \ \ \sigma_1$$

$$\boxed{summary_{\overline{W}}\ \ \ \mathbf{:}\ \ \ WithItem\ \ \textbf{list}\ \ \ \rightarrow\ \ \ \Sigma\ \ \rightarrow\ \ \ tl}$$
$$summary_{\overline{W}}[\![\ with\_item_1\ with\_item_2\ ...\ with\_item_n\ ]\!](\sigma) =$$
$$summary_W[\![\ with\_item_1\ ]\!](\sigma) \sqcup\ \ summary_W[\![\ with\_item_2\ ]\!](\sigma) \sqcup\ \ \textbf{...}\ \ \ \ summary_W[\![\ with\_item_n\ ]\!](\sigma)$$

$$\boxed{summary_W\ \ \ \mathbf{:}\ \ \ WithItem\ \ \rightarrow\ \ \ \Sigma\ \ \rightarrow\ \ \ tl}$$
$$summary_W[\![\ expr_1\ \texttt{as}\ \ expr_2?\ ]\!](\sigma) = summary_E[\![\ expr_1\ ]\!](\sigma)$$

$$\boxed{summary_E\ \ \ \mathbf{:}\ \ \ Expr\ \ \rightarrow\ \ \ \Sigma\ \ \ \ \rightarrow\ \ \ tl}$$

A strict form of call expression

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

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