Code Transformation for Distributed Python ML Code

1 Python Abstract Syntax

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

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

$match\_case$	$::=$	$\texttt{case}\ \ pattern\ \ (\texttt{if}\ \ expr)?\ \texttt{:}\ \ stmt^*$	(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)
	$\mid$	$pattern\ \ \texttt{\|}\ pattern$	(MatchOr)
	$\mid$	$\_$	(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)
	$\mid$	$\ldots$	(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

τ	$::=$	$Module$	(Modules)
		$Stmt$	(Statements)
		$Expr$	(Expressions)
		$Comprehension$	(Comprehensions)
		$ExcHandler$	(Exception handlers)
		$Alias$	(Aliases)
		$WithItem$	(Withitems)
		$Pattern$	(Patterns)
		Id	(Identifiers)
		Str	(Strings)
		τ list	(List of elements typed τ)

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

$\mathbf{.\_1}$	:	$\tau \times \tau \rightarrow \tau$	Get the first element of the given pair
$\mathbf{@}$	:	τ list $\rightarrow \tau$ list $\rightarrow \tau$ list	Concatenate two lists
$\mathbf{::}$	:	$\tau \rightarrow \tau$ list $\rightarrow \tau$ list	Append an element to a list (right-associative)

2.2.2 Transformation Rules

$$\boxed{trans_M \quad : \quad Module \ \rightarrow \ Module}$$

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

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

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

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

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

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

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

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

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

A strict form of assignment statements

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

IF σ(**"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)?],

σ[**"dataset"**$\mapsto$ id_r])

ELIF σ(**"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)?],

σ[**"checkpoint"**$\mapsto$ id_r])

ELIF σ(**"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)?], 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** $1 \leq \mathrm{i} \leq \mathrm{k}$ **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]], 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])

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)

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

$$\begin{array}{l}
trans_S[\![\ expr_1\ \texttt{:}\ expr_2\ (\texttt{=}\ expr_3)?\ ]\!](\sigma) = ([expr_1\ \texttt{:}\ expr_2\ (\texttt{=}\ trans_E[\![\ expr_3\ ]\!](\sigma))?],\ \sigma)\\
\quad \textbf{IF}\ expr_1 = id_1\ \ \textbf{AND}\ \sigma(\textbf{``tensor_flow''}) = id_2\\
\quad \textbf{AND}\ expr_3 = id_2\texttt{.data.Dataset.}expr_4\ \texttt{(}expr_5^*\ keyword^*\texttt{)}\ \textbf{THEN}\\
\qquad ([expr_1\ \texttt{:}\ expr_2\ \texttt{=}\ expr_3\ (\texttt{\#type:}s)?],\ \sigma[\textbf{``dataset''} \mapsto id_1])\\
\quad \textbf{ELSE}\ ([expr_1\ \texttt{:}\ expr_2\ (\texttt{=}\ trans_E[\![\ expr_3\ ]\!](\sigma))?],\ \sigma)
\end{array}$$

$$\begin{array}{l}
trans_S[\![\ (\texttt{\#type:}s)?\ \texttt{for}\ expr_1\ \texttt{in}\ expr_2\ \texttt{:}\ stmt_1^*\ (\texttt{else}\ \texttt{:}\ stmt_2^*)?\ ]\!](\sigma) =\\
\quad ([(\texttt{\#type:}s)?\ \texttt{for}\ expr_1\ \texttt{in}\ trans_E[\![\ expr_2\ ]\!](\sigma)\ \texttt{:}\\
\qquad trans_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)._\mathbf{1}\ (\texttt{else}\ \texttt{:}\ trans_{\overline{S}}[\![\ stmt_2^*\ ]\!](\sigma)._\mathbf{1})?],\ \sigma)
\end{array}$$

$$\begin{array}{l}
trans_S[\![\ (\texttt{\#type:}s)?\ \texttt{async for}\ expr_1\ \texttt{in}\ expr_2\ \texttt{:}\ stmt_1^*\ (\texttt{else}\ \texttt{:}\ stmt_2^*)?\ ]\!](\sigma) =\\
\quad ([(\texttt{\#type:}s)?\ \texttt{async for}\ expr_1\ \texttt{in}\ trans_E[\![\ expr_2\ ]\!](\sigma)\ \texttt{:}\\
\qquad trans_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)._\mathbf{1}\ (\texttt{else}\ \texttt{:}\ trans_{\overline{S}}[\![\ stmt_2^*\ ]\!](\sigma)._\mathbf{1})?],\ \sigma)
\end{array}$$

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

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

$$\begin{array}{l}
trans_S[\![\ (\texttt{\#type:}s)?\ \texttt{with}\ with_item^*\ \texttt{:}\ stmt^*\ ]\!](\sigma) =\\
\quad \textbf{LET}\ with_item^{*\prime}, \sigma_1 = trans_{\overline{W}}[\![\ with_item^*\ ]\!](\sigma)\ \textbf{IN}\\
\quad \textbf{LET}\ stmt^{*\prime}, \sigma_2 = trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma_1)\ \textbf{IN}\\
\quad \textbf{IF}\ \sigma_1 \setminus \sigma = [\textbf{``gradient_tape''} \mapsto id]\ \ \textbf{THEN}\\
\qquad ([(\texttt{\#type:}s)?\ \texttt{with}\ with_item^{*\prime}\ \texttt{:}\ stmt^{*\prime},\\
\qquad id\ \texttt{= hvd.DistributedGradientTape(}id\texttt{)}],\ \sigma_2)\\
\quad \textbf{ELSE}\ ([(\texttt{\#type:}s)?\ \texttt{with}\ with_item^{*\prime}\ \texttt{:}\ stmt^{*\prime}],\ \sigma_2)
\end{array}$$

$$\begin{array}{l}
trans_S[\![\ (\texttt{\#type:}s)?\ \texttt{async with}\ with_item^*\ \texttt{:}\ stmt^*\ ]\!](\sigma) =\\
\quad \textbf{LET}\ with_item^{*\prime}, \sigma_1 = trans_{\overline{W}}[\![\ with_item^*\ ]\!](\sigma)\ \textbf{IN}\\
\quad \textbf{LET}\ stmt^{*\prime}, \sigma_2 = trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma_1)\ \textbf{IN}\\
\quad \textbf{IF}\ \sigma_1 \setminus \sigma = [\textbf{``gradient_tape''} \mapsto id]\ \ \textbf{THEN}\\
\qquad ([(\texttt{\#type:}s)?\ \texttt{async with}\ with_item^{*\prime}\ \texttt{:}\ stmt^{*\prime},\\
\qquad id\ \texttt{= hvd.DistributedGradientTape(}id\texttt{)}],\ \sigma_2)\\
\quad \textbf{ELSE}\ ([(\texttt{\#type:}s)?\ \texttt{async with}\ with_item^{*\prime}\ \texttt{:}\ stmt^{*\prime}],\ \sigma_2)
\end{array}$$

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

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

$trans_S[\![$ `try : ` $stmt_1^*$ $exc\_handler^*$ (`else : ` $stmt_2^*$)? (`finally : ` $stmt_3^*$)? $]\!](\sigma) =$
([`try : ` $trans_{\overline{S}}[\![\ stmt_1^*\ ]\!](\sigma)$.**_1** $(trans_H[\![exc\_handler]\!](\sigma))^*$
(`else : ` $trans_{\overline{S}}[\![\ stmt_2^*\ ]\!](\sigma)$.**_1**)? (`finally : ` $trans_{\overline{S}}[\![\ stmt_3^*\ ]\!](\sigma)$.**_1**)?], σ)

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

$trans_S[\![$ `import` $alias^*\ ]\!](\sigma) =$
LET $\sigma_1 = trans_{\overline{A}}[\![\ alias^*\ ]\!](\sigma)$ **IN**
IF $\sigma_1 \setminus \sigma = [$**"tensor_flow"**$\mapsto id]$ **THEN**
([`import` $alias^*$,
`import horovod.tensorflow as hvd`,
`hvd_broadcast_done = False`,
`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 ([`import` $alias^*$], σ_1)

$trans_S[\![$ `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)

$trans_S[\![$ `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) ⟧(σ) =
  IF σ("optimizer") = id_t  AND expr_1 = id_t.apply_gradients  THEN
    IF id_i = grads_and_vars  WHEN 1 ≤ i ≤ 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  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 = callbacks  WHEN 1 ≤ i ≤ k  THEN
      ([expr_1 (expr_11 ... expr_1n   (id_1 = )? expr_21 ...
        id_i = expr_2i + [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]
      ... (id_k = )? expr_2k)], σ)
    ELSE
      ([expr_1 (expr_11 ... expr_1n   (id_1 = )? expr_21 ... (id_k = )? expr_2k
        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 ≤ i ≤ 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)
  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

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

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

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

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

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

$trans_E⟦\ expr_1\ \ boolop\ \ expr_2\ ⟧(\sigma) = trans_E⟦\ expr_1\ ⟧(\sigma)\ \ boolop\ \ trans_E⟦\ expr_2\ ⟧(\sigma)$

$trans_E⟦\ expr_1\ \ \textbf{:=}\ \ expr_2\ ⟧(\sigma) = expr_1\ \ \textbf{:=}\ \ trans_E⟦\ expr_2\ ⟧(\sigma)$

$trans_E⟦\ expr_1\ \ binop\ \ expr_2\ ⟧(\sigma) = trans_E⟦\ expr_1\ ⟧(\sigma)\ \ binop\ \ trans_E⟦\ expr_2\ ⟧(\sigma)$

$trans_E⟦\ unop\ \ expr\ ⟧(\sigma) = unop\ \ trans_E⟦\ expr\ ⟧(\sigma)$

$trans_E⟦\ \texttt{lambda}\ \ args\ \ \texttt{:}\ \ expr\ ⟧(\sigma) = \texttt{lambda}\ \ args\ \ \texttt{:}\ \ trans_E⟦\ expr\ ⟧(\sigma)$

$trans_E⟦\ expr_1\ \ \texttt{if}\ \ expr_2\ \ \texttt{else}\ \ expr_3\ ⟧(\sigma) =$
$\quad trans_E⟦\ expr_1\ ⟧(\sigma)\ \ \texttt{if}\ \ trans_E⟦\ expr_2\ ⟧(\sigma)\ \ \texttt{else}\ \ trans_E⟦\ expr_3\ ⟧(\sigma)$

$trans_E⟦\ expr_1\ \ \texttt{if}\ \ expr_2\ \ \texttt{else}\ \ expr_3\ ⟧(\sigma) =$
$\quad trans_E⟦\ expr_1\ ⟧(\sigma)\ \ \texttt{if}\ \ trans_E⟦\ expr_2\ ⟧(\sigma)\ \ \texttt{else}\ \ trans_E⟦\ expr_3\ ⟧(\sigma)$

$trans_E⟦\ \{(expr_1\ :\ expr_2)^*\ \ (**expr_3)^*\}\ ⟧(\sigma) = \{(expr_1\ :\ trans_E⟦expr_2⟧(\sigma))^*\ \ (**expr_3)^*\}$

$trans_E⟦\ \{expr^*\}\ ⟧(\sigma) = \{(trans_E⟦expr⟧(\sigma))^*\}$

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

$trans_E⟦\ \texttt{(}expr^*\texttt{)}\ ⟧(\sigma) = \texttt{(}(trans_E⟦expr⟧(\sigma))^*\texttt{)}$

$trans_E⟦\ \{expr_1\texttt{:}expr_2\ \ comprehension^*\}\ ⟧(\sigma) = \{expr_1\texttt{:}trans_E⟦\ \ expr_2\ \ ⟧(\sigma)\ \ (trans_O⟦comprehension⟧(\sigma))^*\}$

$trans_E⟦\ \{expr\ \ \ comprehension^*\}\ ⟧(\sigma) = \{trans_E⟦\ \ expr\ \ ⟧(\sigma)\ \ (trans_O⟦comprehension⟧(\sigma))^*\}$

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

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

$trans_E[\![$ `await` $expr\ ]\!](\sigma) =$ `await` $trans_E[\![\ expr\ ]\!](\sigma)$

$trans_E[\![$ `yield` $expr?\ ]\!](\sigma) =$ `yield` $(trans_E[\![\ expr\ ]\!](\sigma))?$

$trans_E[\![$ `yield from` $expr\ ]\!](\sigma) =$ `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 $\sigma($**"dataset"**$) = id_t$ **AND** $expr_1 = id_t$`.take` **THEN**
IF $id_i =$ `count` **WHEN** $1 \leq \mathrm{i} \leq \mathrm{k}$ **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**

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

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

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

$trans_A[\![$ 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"**$\mapsto id]$
ELSE σ

$trans_A[\![$ id_1 `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"**$\mapsto id_2]$
ELSE σ

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

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

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

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

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

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

$trans_W[\![$ $expr_1$ `as` $expr_2$ $]\!]$(smodenv) =

IF $\sigma($**"tensor_flow"**$)$ = id_1 **AND** $expr_1$ = id_1`.GradientTape()` **AND** $expr_2$ = id_2 **THEN**
$(expr_1$ `as` $expr_2, \sigma[$**"gradient_tape"**$\mapsto id_2])$
ELSE $(trans_E[\![$ $expr_1$ $]\!](\sigma)$ `as` $expr_2, \sigma)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\sigma \ \in \Sigma \ = Id \overset{\text{fin}}{\to} Summary$ ENVIRONMENT STORING MAPPINGS FROM IDS TO SUMMARIES

3.2.2 Training Loop

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

$summary_M[\![\ id \ \ stmt^* \ \ type\_ignore \ ]\!] =$

$\quad \textbf{LET} \ \sigma, tl \ = \ summary_{\overline{S}}[\![\ stmt^* \ ]\!](\sigma) \ \ \textbf{IN}$

$\quad \texttt{ModuleSummary} \ \ id \ \ \sigma \ \ tl$

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

$summary_{\overline{S}}[\![\ stmt_1 \ \ stmt_2 \ ... \ stmt_n \ ]\!](\sigma) \ = \ \textbf{LET} \ \sigma_1, tl_1 \ = \ summary_S[\![\ stmt_1 \ ]\!](\sigma) \ \ \textbf{IN}$

$\textbf{LET} \ \sigma_2, tl_2 \ = \ summary_{\overline{S}}[\![\ stmt_2 \ ]\!](\sigma_1) \ \ \textbf{IN}$

$\dots$

$\textbf{LET} \ \sigma_n, tl_n \ = \ summary_{\overline{S}}[\![\ stmt_n \ ]\!](\sigma_{n-1}) \ \ \textbf{IN}$

$(\sigma_n, tl_1 \sqcup tl_2 \sqcup \ \dots \ tl_n)$

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

$summary_S[\![\ (@expr_1)^* \ \ \texttt{def} \ \ id \ \ \texttt{(}args\texttt{)} \ \ (\texttt{->} \ expr_2)? \ \ : \ (\texttt{\#type:}s)? \ \ stmt^* \ ]\!](\sigma) =$

$\quad \textbf{LET} \ \sigma', tl \ = summary_{\overline{S}}[\![\ stmt^* \ ]\!] \ \ \textbf{IN}$

$\quad (\sigma[id \mapsto \texttt{FuncSummary} \ \ tl], \bot)$

$summary_S[\![\ (@expr_1)^* \ \ \texttt{async def} \ \ id \ \ \texttt{(}args\texttt{)} \ \ (\texttt{->} \ expr_2)? \ \ : \ (\texttt{\#type:}s)? \ \ stmt^* \ ]\!](\sigma) =$

$\quad \textbf{LET} \ \sigma', tl \ = summary_{\overline{S}}[\![\ stmt^* \ ]\!] \ \ \textbf{IN}$

$\quad (\sigma[id \mapsto \texttt{FuncSummary} \ \ tl], \bot)$

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

$\quad \textbf{IF} \ \ expr_{1i} \ = \ id_c\texttt{.keras.Model} \ \ \textbf{WHEN} \ \ 1 \le \text{i} \le \text{n} \ \ \textbf{AND}$

$\qquad \sigma(id_c) \ = \ \texttt{ModuleSummary} \ \ \text{Id(”tensorflow”)} \ \ [] \ \ \bot \ \textbf{THEN} \ \ (\sigma[id \mapsto \texttt{ClassSummary} \ \ Model], \bot)$

$\quad \textbf{ELIF} \ \ expr_{2i} \ = \ id_c\texttt{.keras.Model} \ \ \textbf{WHEN} \ \ 1 \le \text{i} \le \text{k} \ \ \textbf{AND}$

$\qquad \sigma(id_c) \ = \ \texttt{ModuleSummary} \ \ \text{Id(”tensorflow”)} \ \ [] \ \ \bot \ \textbf{THEN} \ \ (\sigma[id \mapsto \texttt{ClassSummary} \ \ Model], \bot)$

$\quad \textbf{ELSE} \ \ (\sigma[id \mapsto \texttt{ClassSummary} \ \ \bot], \bot)$

A strict form of assignment statements

$summary_S[\![\ id_r \ \texttt{=} \ id_c \ \texttt{()} \ (\texttt{\#type:}s)? \ ]\!](\sigma) =$

$\quad \textbf{IF} \ \ \sigma(id_c) \ = \ \texttt{ClassSummary} \ \ Model \ \ \textbf{THEN} \ \ (\sigma[id_r \mapsto \texttt{ValueSummary} \ \ \textbf{”model\_instance”}], \bot)$

$\quad \textbf{ELSE} \ \ (\sigma, \bot)$

$summary_S[\![\ (\texttt{\#type:}s)? \ \ \texttt{for} \ \ expr_1 \ \ \texttt{in} \ \ expr_2 \ \ : \ \ stmt_1^* \ \ (\texttt{else} \ : \ \ stmt_2^*)? \ ]\!](\sigma) =$

$\quad \textbf{LET} \ \ \_, tl \ = \ summary_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma) \ \ \textbf{IN}$

$\quad (\sigma, tl)$

$summary_S[\![\ (\texttt{\#type:}s)? \ \ \texttt{async for} \ \ expr_1 \ \ \texttt{in} \ \ expr_2 \ \ : \ \ stmt_1^* \ \ (\texttt{else} \ : \ \ stmt_2^*)? \ ]\!](\sigma) =$

$\quad \textbf{LET} \ \ \_, tl \ = \ summary_{\overline{S}}[\![\ stmt_1^* \ ]\!](\sigma) \ \ \textbf{IN}$

$\quad (\sigma, tl)$

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

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

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

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

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

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

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

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

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

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

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

$\boxed{summary_W \quad : \quad WithItem \ \rightarrow \ \Sigma \ \rightarrow \ tl}$

$summary_W[\![\ expr_1$ `as` $expr_2?\ ]\!](\sigma) = summary_E[\![\ expr_1\ ]\!](\sigma)$

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

A strict form of call expression

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

IF $\sigma(id_r) =$ `ValueSummary` **"model_instance"** **AND**

$expr_1 = id_r$`.fit` **THEN** *Optimizer*

ELIF $\sigma(id_r) =$ `ModuleSummary` Id("tensorflow") σ' tl **AND**

$expr_1 = id_r$`.GradientTape` **THEN** *GradTape*

ELIF $\sigma(id_r) =$ `FuncSummary` tl **AND**

$expr_1 =$ `idr` **THEN** tl

ELSE $\perp$

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