Code Transformation for Distributed Python ML Code

1 Python Abstract Syntax

module	::=	*stmt** *type_ignore*	(MODULEDEF)
stmt	::=	(@*expr*)* def *id* (*args*) (-> *expr*)? : (#type:*s*)? *stmt**	(FUNDEF)
	|	(@*expr*)* async def *id* (*args*) (-> *expr*)? : (#type:*s*)? *stmt**	(ASYNCFUNDEF)
	|	(@*expr*)* class *id* (*expr** *keyword**) : *stmt**	(CLASSDEF)
	|	return *expr*?	(RETURN)
	|	delete *expr**	(DELETE)
	|	*expr** = *expr* (#type:*s*)?	(ASSIGN)
	|	*expr* *binop* = *expr*	(AUGASSIGN)
	|	*expr* : *expr* (= *expr*)?	(ANNASSIGN)
	|	(#type:*s*)? for *expr* in *expr* : *stmt** (else : *stmt**)?	(FORLOOP)
	|	(#type:*s*)? async for *expr* in *expr* : *stmt** (else : *stmt**)?	(ASYNCFORLOOP)
	|	while (*expr*) : *stmt** (else : *stmt**)?	(WHILELOOP)
	|	if (*expr*) : *stmt** (else : *stmt**)?	(IF)
	|	(#type:*s*)? with *with_item** : *stmt**	(WITH)
	|	(#type:*s*)? async with *with_item** : *stmt**	(ASYNCWITH)
	|	match *expr* : *match_case**	(MATCH)
	|	raise *expr*? (from *expr*)?	(RAISE)
	|	try : *stmt** *exc_handler** (else : *stmt**)? (finally : *stmt**)?	(TRY)
	|	assert *expr* *expr*?	(ASSERT)
	|	import *alias**	(IMPORT)
	|	from *i* *id*? import *alias**	(IMPORTFROM)
	|	global *id**	(GLOBAL)
	|	nonlocal *id**	(NONLOCAL)
	|	*expr*	(EXPRSTMT)
	|	pass	(PASS)
	|	break	(BREAK)
	|	continue	(CONTINUE)

expr	::=	*expr* *boolop* *expr*	(BOOLOP)
	|	*expr* := *expr*	(NAMEDEXPR)
	|	*expr* *binop* *expr*	(BINARYOP)
	|	*unop* *expr*	(UNARYOP)
	|	**lambda** *args* : *expr*	(LAMBDA)
	|	*expr* **if** *expr* **else** *expr*	(IFEXPR)
	|	{(*expr* : *expr*)* (**expr*)*}	(DICTIONARY)
	|	{*expr**}	(SET)
	|	[*expr**]	(LIST)
	|	(*expr**)	(TUPLE)
	|	{*expr*:*expr* *comprehension**}	(DICTIONARYCOMP)
	|	{*expr* *comprehension**}	(SETCOMP)
	|	[*expr* *comprehension**]	(LISTCOMP)
	|	(*expr* *comprehension**)	(GENERATORCOMP)
	|	**await** *expr*	(AWAIT)
	|	**yield** *expr*?	(YIELD)
	|	**yield from** *expr*	(YIELDFROM)
	|	*expr* (*compop* *expr*)*	(COMPOP)
	|	*expr* (*expr** *keyword**)	(CALL)
	|	{*expr* (!*i*)? (:*expr*)?}	(FORMATTEDVALUE)
	|	*expr**	(JOINEDSTR)
	|	*constant*	(CONSTANT)
	|	*expr*.*id*	(ATTRIBUTE)
	|	*expr*[*expr*]	(SUBSCRIPT)
	|	*expr*	(STARRED)
	|	**expr*	(DOUBLESTARRED)
	|	*id*	(NAME)
	|	*expr*? (:*expr*)? (:*expr*)?	(SLICE)
boolop	::=	**and** | **or**	(BOOLOPERATOR)
binop	::=	+ | - | * | @ | / | ** | << | >> | | | ^ | & | // | %	(BINOPERATOR)
unop	::=	~ | **not** | + | -	(UNOPERATOR)
compop	::=	== | != | < | <= | > | >= | **is** | **is not** | **in** | **not in**	(COMPOPERATOR)
comprehension	::=	**for** *expr* **in** *expr* (**if** *expr*)*	(COMPREHENSION)
	|	**async for** *expr* **in** *expr* (**if** *expr*)*	(ASYNCCOMPREHEN)
exc_handler	::=	**except** *expr*? (**as** *id*)? : *stmt**	(EXCHANDLER)
args	::=	(*arg* (= *expr*)?)*, (*arg* (= *expr*)?)*, *arg*?, (*arg* (= *expr*)?)*, *arg*?	(ARGUMENTS)
arg	::=	*id* *expr*? *s*?	(ARGUMENT)
keyword	::=	*id*? = *expr*	(KEYWORD)
alias	::=	*id* (.*id*)* (**as** *id*)?	(ALIAS)
with_item	::=	*expr* (**as** *expr*)?	(WITHITEM)

2

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

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

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

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

$$
\begin{array}{llll}
\textbf{.\_1} & : & \tau \times \tau \to \tau & \text{Get the first element of the given pair} \\
\textbf{@} & : & \tau \text{ list} \to \tau \text{ list} \to \tau \text{ list} & \text{Concatenate two lists} \\
\textbf{::} & : & \tau \to \tau \text{ list} \to \tau \text{ list} & \text{Append an element to a list (right-associative)}
\end{array}
$$

2.2.2 Transformation Rules

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

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

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

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

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

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

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

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

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

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

A strict form of assignment statements

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

$$\ldots\ (id_k\ =\ )?\ expr_{2k})\ (\text{\#type}:s)?],\ \sigma[\text{``\textbf{optimizer}''}\mapsto id_r])$$

ELSE

$$([id_r\ =\ expr_1\ (expr_{11}\ *\ \text{hvd.size()}\ \ldots\ expr_{1n}\quad (id_1\ =\ )?\ expr_{21}\ \ldots\ (id_k\ =\ )?\ expr_{2k})\ (\text{\#type}:s)?],$$
$$\sigma[\text{``\textbf{optimizer}''}\mapsto id_r])$$

ELIF $\sigma(\text{``\textbf{optimizer}''})\ =\ id_t$ **AND** $expr_1\ =\ id_t.\text{apply\_gradients}$ **THEN**

  IF $id_i\ =\ \text{grads\_and\_vars}$ **WHEN** $1 \leq i \leq k$ **THEN**

    LET $id_z\ =\ \textbf{NewID()}$ **IN**

$$([id_z\ =\ expr_{2i},$$
$$id_r\ =\ expr_1\ (expr_{11}\ \ldots\ expr_{1n}\quad (id_1\ =\ )?\ expr_{21}\ \ldots\ id_i\ =\ id_z\ \ldots\ (id_k\ =\ )?\ expr_{2k})\ (\text{\#type}:s)?,$$

```
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 $id_z\ =\ \textbf{NewID()}$ **IN**

$$([id_z\ =\ expr_{11},$$
$$id_r\ =\ expr_1\ (id_z\ expr_{12}\ \ldots\ expr_{1n}\quad (id_1\ =\ )?\ expr_{21}\ \ldots\ (id_k\ =\ )?\ expr_{2k})\ (\text{\#type}:s)?,$$

```
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 ]],
```
$\sigma)$

ELIF $\sigma(\text{``\textbf{checkpoint}''})\ =\ id_t$ **AND** $expr_1\ =\ id_t.\text{save}$ **THEN**

$$([\text{if hvd.rank() == 0:}\ [id_r\ =\ expr_1\ (expr_{11}\ \ldots\ expr_{1n}\quad (id_1\ =\ )?\ expr_{21}\ \ldots\ (id_k\ =\ )?\ expr_{2k})\ (\text{\#type}:s)?]],$$
$$\sigma)$$

ELIF $\sigma(\text{``\textbf{keras}''})\ =\ id_k$ **AND** $expr_1\ =\ id_k.\text{models.Sequential}$ **THEN**

$$([id_r\ =\ expr_1\ (expr_{11}\ \ldots\ expr_{1n}\quad (id_2\ =\ )?\ expr_{21}\ \ldots\ (id_k\ =\ )?\ expr_{2k})],\ \sigma[\text{``\textbf{model}''}\mapsto id_r])$$

ELIF $\sigma(\text{``\textbf{keras}''})\ =\ id_t$ **AND** $expr_1\ =\ id_t.\text{optimizers.Adam}$ **THEN**

  IF $id_i\ =\ \text{learning\_rate}$ **WHEN** $1 \leq i \leq k$ **THEN**

$$([id_r\ =\ expr_1\ (expr_{11}\ \ldots\ expr_{1n}\quad (id_1\ =\ )?\ expr_{21}\ \ldots\ id_i\ =\ expr_{2i}\ *\ \text{hvd.size()}$$
$$\ldots\ (id_k\ =\ )?\ expr_{2k})\ (\text{\#type}:s)?$$
$$id_r\ =\ \text{hvd.DistributedOptimizer}(id_r)],\ \sigma[\text{``\textbf{optimizer}''}\mapsto id_r])$$

  ELSE

$$([id_r\ =\ expr_1\ (expr_{11}\ *\ \text{hvd.size()}\ \ldots\ expr_{1n}\quad (id_1\ =\ )?\ expr_{21}\ \ldots\ (id_k\ =\ )?\ expr_{2k})\ (\text{\#type}:s)?$$
$$id_r\ =\ \text{hvd.DistributedOptimizer}(id_r)],\ \sigma[\text{``\textbf{optimizer}''}\mapsto id_r])$$

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

$trans_S[\![\ expr_0\ =\ expr_1\ (\text{\#type}:s)?\ ]\!](\sigma) =$
  IF $\sigma(\text{``\textbf{os}''})\ =\ id_{os}$ **AND** $expr_0\ =\ id_{os}.\text{environ ['CUDA\_VISIBLE\_DEVICES']}$
  THEN $([],\ \sigma)$

A general form of assignment statements except for the strict form
$trans_S[\![\ expr_1^*\ =\ expr_2\ (\text{\#type}:s)?\ ]\!](\sigma) = ([expr_1^*\ =\ trans_E[\![\ expr_2\ ]\!](\sigma)\ (\text{\#type}:s)?],\ \sigma)$

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

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

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

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

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

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

$trans_S[\![\ (\texttt{\#type:}s)?\ \texttt{with}\ with\_item^* : stmt^*\ ]\!](\sigma) =$
    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 $\sigma_1 \setminus \sigma = [\textbf{``gradient\_tape"} \mapsto id]$ **THEN**
       $([(\texttt{\#type:}s)?\ \texttt{with}\ with\_item^{*\prime} : stmt^{*\prime},$
       $id\ \texttt{=}\ \texttt{hvd.DistributedGradientTape}(id)], \sigma_2)$
    ELSE $([(\texttt{\#type:}s)?\ \texttt{with}\ with\_item^{*\prime} : stmt^{*\prime}], \sigma_2)$

$trans_S[\![\ (\texttt{\#type:}s)?\ \texttt{async with}\ with\_item^* : stmt^*\ ]\!](\sigma) =$
    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 $\sigma_1 \setminus \sigma = [\textbf{``gradient\_tape"} \mapsto id]$ **THEN**
       $([(\texttt{\#type:}s)?\ \texttt{async with}\ with\_item^{*\prime} : stmt^{*\prime},$
       $id\ \texttt{=}\ \texttt{hvd.DistributedGradientTape}(id)], \sigma_2)$
    ELSE $([(\texttt{\#type:}s)?\ \texttt{async with}\ with\_item^{*\prime} : stmt^{*\prime}], \sigma_2)$

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

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

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

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

$trans_S[\![$ 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')], `$\sigma_1)$
    ELSE $([$import $\ alias^*], \sigma_1)$

$trans_S[\![$ 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.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')], `$\sigma_1)$
    ELSE $([$import $\ alias^*], \sigma_1)$

$trans_S[\![$ 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')], `$\sigma_1)$
    ELSE $([$from $\ i \ id?$ import $\ alias^*], \sigma_1)$

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

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

A strict form of expr statements

$trans_S[\![\ expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 \ \texttt{=} \ )? \ expr_{21} \ ... \ (id_k \ \texttt{=} \ )? \ expr_{2k}) \ ]\!](\sigma) =$
    IF $\ \sigma($**"optimizer"**$) \ = \ id_t$ **AND** $\ expr_1 \ = \ id_t$`.apply_gradients`   **THEN**

IF id_i = `grads_and_vars` **WHEN** $1 \leq i \leq k$ **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` $]], \sigma)$

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` $]], \sigma)$

ELIF $expr_1$ = `print` **THEN**

  $([$`if hvd.rank() == 0:`,

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

ELIF $\sigma(\text{``checkpoint''})$ = id_t **AND** $expr_1$ = id_t`.save` **THEN**

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

ELIF $\sigma(\text{``model''})$ = id_t **AND** $expr_1$ = id_t`.fit` **THEN**

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

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

      $([expr_1$ $(expr_{11}$ ... $expr_{1n}$ $(id_1$ = $)?$ $expr_{21}$ ...

        id_i `= 1 if hvd.rank() == 0 else 0`

        id_j = $expr_{2j}$ `+ [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]`

      ... $(id_k$ = $)?$ $expr_{2k})], \sigma)$

    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 =` $expr_{2j}$ `+ [hvd.callbacks.BroadcastGlobalVariablesCallback(0)])], \sigma)`

  ELSE

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

      $([expr_1$ $(expr_{11}$ ... $expr_{1n}$ $(id_1$ = $)?$ $expr_{21}$ ...

        id_j = $expr_{2j}$ `+ [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]` ... $(id_k$ = $)?$ $expr_{2k}$

        `verbose = 1 if hvd.rank() == 0 else 0)]`, $\sigma)$

    ELSE

      $([expr_1$ $(expr_{11}$ ... $expr_{1n}$ $(id_1$ = $)?$ $expr_{21}$ ... $(id_k$ = $)?$ $expr_{2k}$

        `verbose = 1 if hvd.rank() == 0 else 0`

        `callbacks =` $expr_{2j}$ `+ [hvd.callbacks.BroadcastGlobalVariablesCallback(0)])]`, $\sigma)$

ELIF $\sigma(\text{``model''})$ = id_t **AND** $expr_1$ = id_t`.compile` **THEN**

  IF id_i = `optimizer` **AND** $expr_{2i}$ = `"adam"` **WHEN** $1 \leq i \leq k$ **THEN**

    $([id_z$ = `tf.optimizers.Adam(learning_rate=0.001 * hvd.size())`

$$id_z \quad = \quad \texttt{hvd.DistributedOptimizer}(id_z)$$
$$expr_1 \ (expr_{11} \ ... \ expr_{1n} \quad (id_1 \ \texttt{=} \ )? \ expr_{21} \ ... \ id_i \ \texttt{=} \ id_z \ ... \ (id_k \ \texttt{=} \ )? \ expr_{2k})], \sigma)$$

ELIF $expr_{11} \quad = \quad$ `"adam"` **THEN**
$$([id_z \quad = \quad \texttt{tf.optimizers.Adam(learning\_rate=0.001 * hvd.size())}$$
$$id_z \quad = \quad \texttt{hvd.DistributedOptimizer}(id_z)$$
$$expr_1 \ (id_z \ ... \ expr_{1n} \quad (id_1 \ \texttt{=} \ )? \ expr_{21} \ ... \ (id_k \ \texttt{=} \ )? \ expr_{2k})], \sigma)$$

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

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

A general form of expr statements except for the strict form

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

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

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

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

$$\boxed{trans_E \quad : \quad Expr \quad \to \quad \Sigma \quad \to \quad Expr}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$trans_E[\![\ expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1 =)?\ expr_{21}\ ...\ (id_k =)?\ expr_{2k})\ ]\!](\sigma) =$
    IF $\sigma(\text{"dataset"}) = id_t$ **AND** $expr_1 = id_t.\texttt{take}$ **THEN**
        IF $id_i = \texttt{count}$ **WHEN** $1 \leq i \leq k$ **THEN**
            $expr_1\ (expr_{11}\ ...\ expr_{1n}\quad (id_1 =)?\ expr_{21}\ ...\ id_i = expr_{2i}\ \texttt{// hvd.size()}\ ...\ (id_k =)?\ expr_{2k})$
        ELSE
            $expr_1\ (expr_{11}\ \texttt{// hvd.size()}\ ...\ expr_{1n}\quad (id_1 =)?\ expr_{21}\ ...\ (id_k =)?\ expr_{2k})$
    ELSE
        $trans_E[\![\ expr_1\ ]\!](\sigma)\ (trans_E[\![\ expr_{11}\ ]\!](\sigma)\ ...\ trans_E[\![\ expr_{1n}\ ]\!](\sigma)$
            $(id_1 =)?\ trans_E[\![\ expr_{21}\ ]\!](\sigma)\ ...\ (id_k =)?\ trans_E[\![\ expr_{2k}\ ]\!](\sigma))$

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

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

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

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

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

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

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

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

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

$trans_O \quad : \quad Comprehension \; \rightarrow \; \Sigma \; \rightarrow \; Comprehension$

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

$trans_H \quad : \quad ExcHandler \; \rightarrow \; \Sigma \; \rightarrow \; ExcHandler$

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

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

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

$trans_A \quad : \quad Alias \; \rightarrow \; \Sigma \; \rightarrow \; \Sigma$

$trans_A[\![\; id \;]\!](\sigma) =$
   LET $id = \texttt{tensorflow}$ **THEN** $\sigma[\text{"tensor\_flow"} \mapsto id]$
   LET $id = \texttt{keras}$ **THEN** $\sigma[\text{"keras"} \mapsto id]$
   LET $id = \texttt{os}$ **THEN** $\sigma[\text{"os"} \mapsto id]$
   LET $id = \texttt{optimizers}$ **THEN** $\sigma[\text{"optimizers"} \mapsto id]$
   ELSE σ

$trans_A[\![\; id_1 \; \texttt{as} \; \; id_2 \;]\!](\sigma) =$
   LET $id_1 = \texttt{tensorflow}$ **THEN** $\sigma[\text{"tensor\_flow"} \mapsto id_2]$
   LET $id_1 = \texttt{tensorflow.keras}$ **THEN** $\sigma[\text{"keras"} \mapsto id_2]$
   LET $id_1 = \texttt{os}$ **THEN** $\sigma[\text{"os"} \mapsto id_2]$
   LET $id = \texttt{optimizers}$ **THEN** $\sigma[\text{"optimizers"} \mapsto id]$
   ELSE σ

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

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

$trans_{\overline{W}}[\![\; with\_item_1 \; with\_item_2 \; ... \; with\_item_n \;]\!](\sigma) =$
   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**
   $(with\_item_1' :: with\_item_2' :: ... :: [with\_item_n'], \sigma_n)$

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

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

$trans_W[\![\ expr_1\ \texttt{as}\ \ expr_2\ ]\!](\text{smodenv}) =$
  IF $\sigma(\textbf{``tensor\_flow''}) \;=\; id_1$ **AND** $expr_1 \;=\; id_1.\texttt{GradientTape()}$ **AND** $expr_2 \;=\; id_2$ **THEN**
    $(expr_1\ \texttt{as}\ \ expr_2,\ \sigma[\textbf{``gradient\_tape''} \mapsto id_2])$
  ELSE $(trans_E[\![\ expr_1\ ]\!](\sigma)\ \texttt{as}\ \ expr_2,\ \sigma)$

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

$trans_C[\![\ \texttt{case}\ pattern\ (\texttt{if}\ \ expr)?\ :\ stmt^*\ ]\!](\sigma) =$
  $\texttt{case}\ \ trans_P[\![\ pattern\ ]\!](\sigma)\ \ (\texttt{if}\ \ trans_E[\![\ expr\ ]\!](\sigma))?\ :\ \ trans_{\overline{S}}[\![\ stmt^*\ ]\!](\sigma)\text{.\_}\mathbf{1}$

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2.2 Training Loop

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

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

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

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

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

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

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

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

A strict form of assignment statements

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

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

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

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

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

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

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

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

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

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

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

$summary_{\overline{A}}\ :\ Alias$ **list** $\to\ \Sigma\ \to\ \Sigma$

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

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

$trans_A[\![\; id_1 \; id_2 \; ... \; id_n \;]\!](\sigma) =$

    LET $\;\sigma_n = [id_n \mapsto \texttt{ModuleSummary} \;\; id_n \quad \top \quad \bot]$

    LET $\;\sigma_{n-1} = [id_{n-1} \mapsto \texttt{ModuleSummary} \;\; id_{n-1} \quad \sigma_n \quad \bot]$

    ...

    LET $\;\sigma_2 = [id_2 \mapsto \texttt{ModuleSummary} \;\; id_2 \quad \sigma_3 \quad \bot]$

    LET $\;\sigma_1 = [id_1 \mapsto \texttt{ModuleSummary} \;\; id_1 \quad \sigma_2 \quad \bot]$

    $\sigma \;\;++\;\; \sigma_1$

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

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

    $summary_W[\![\; with\_item_1 \;]\!](\sigma) \sqcup \;\; summary_W[\![\; with\_item_2 \;]\!](\sigma) \sqcup \;\; \textbf{...} \quad summary_W[\![\; with\_item_n \;]\!](\sigma)$

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

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

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

A strict form of call expression

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

    IF $\; \sigma(id_r) \;=\; \texttt{ValueSummary} \;\; \text{"\textbf{model\_instance}"} \quad$ **AND**

        $expr_1 \;=\; id_r.\texttt{fit} \quad$ **THEN** $\; Optimizer$

    ELIF $\; \sigma(id_r) \;=\; \texttt{ModuleSummary} \; \text{Id("tensorflow")} \;\; \sigma' \quad tl \;$ **AND**

        $expr_1 \;=\; id_r.\texttt{GradientTape} \quad$ **THEN** $\; GradTape$

    ELIF $\; \sigma(id_r) \;=\; \texttt{FuncSummary} \; tl \;$ **AND**

        $expr_1 \;=\; \texttt{idr} \quad$ **THEN** $\; tl$

    ELSE $\; \bot$

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