Package: stablehlo (via r-universe)

June 3, 2026

Title Write stableHLO programs

Version 0.3.0

Description The package offers a low level interface to create stableHLO programs. These programs can be compiled and run on different hardware backends (CPU, GPU, ...) using the 'pjrt' package.

License MIT + file LICENSE

URL https://r-xla.github.io/stablehlo/, https://github.com/r-xla/stablehlo

BugReports https://github.com/r-xla/stablehlo/issues

Imports checkmate, cli, methods, rlang, tengen (>= 0.2.0), withr, xlamisc (>= 0.2.0)

Suggests rmarkdown, knitr, pjrt (>= 0.4.0), testthat (>= 3.0.0)

Additional_repositories https://r-xla.r-universe.dev

Config/build/compilation-database true

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Collate 'aaa.R' 'assert.R' 'conditions.R' 'repr.R' 'shape.R' 'types.R' 'constant.R' 'format_double.R' 'value_id.R' 'func.R' 'func_value.R' 'type_inference.R' 'hlo.R' 'module.R' 'utils.R' 'op.R' 'op-abs.R' 'op-acos.R' 'op-acosh.R' 'op-add.R' 'op-after_all.R' 'op-and.R' 'op-asin.R' 'op-asinh.R' 'op-atan.R' 'op-atan2.R' 'op-atanh.R' 'op-bessel_i1e.R' 'op-bitcast_convert.R' 'op-broadcast_in_dim.R' 'op-call.R' 'op-case.R' 'op-cbrt.R' 'op-ceil.R' 'op-cholesky.R' 'op-clamp.R' 'op-compare.R' 'op-concatenate.R' 'op-constant.R' 'op-convert.R' 'op-cosh.R' 'op-cosine.R' 'op-count_leading_zeros.R' 'op-custom_call.R' 'op-digamma.R'

'op-divide.R' 'op-dot_general.R' 'op-dynamic_slice.R'
'op-dynamic_update_slice.R' 'op-erf.R' 'op-erf_inv.R'
'op-erfc.R' 'op-exponential.R' 'op-exponential_minus_one.R'
'op-floor.R' 'op-gather.R' 'op-if.R' 'op-iota.R'
'op-is_finite.R' 'op-is_inf.R' 'op-is_neg_inf.R'
'op-is_pos_inf.R' 'op-lgamma.R' 'op-log.R' 'op-log_plus_one.R'
'op-logistic.R' 'op-maximum.R' 'op-minimum.R' 'op-multiply.R'
'op-negate.R' 'op-not.R' 'op-or.R' 'op-pad.R' 'op-polygamma.R'
'op-popcnt.R' 'op-power.R' 'op-reduce.R' 'op-reduce_window.R'
'op-remainder.R' 'op-reshape.R' 'op-return.R' 'op-reverse.R'
'op-rng_bit_generator.R' 'op-round_nearest_afz.R'
'op-round_nearest_even.R' 'op-rsqrt.R' 'op-scatter.R'
'op-select.R' 'op-shift_left.R' 'op-shift_right_arithmetic.R'
'op-shift_right_logical.R' 'op-sign.R' 'op-sine.R' 'op-sinh.R'
'op-slice.R' 'op-sort.R' 'op-sqrt.R' 'op-square.R'
'op-subtract.R' 'op-tan.R' 'op-tanh.R' 'op-top_k.R'
'op-transpose.R' 'op-triangular_solve.R' 'op-while.R'
'op-xor.R' 'reexports.R' 'roxygen.R' 'zzz.R'

Config/Needs/website rmarkdown

Repository https://r-xla.r-universe.dev

Date/Publication 2026-06-03 11:42:01 UTC

RemoteUrl https://github.com/r-xla/stablehlo

RemoteRef v0.3.0

RemoteSha 176a83f723d164e8b8b214ef89e035e24f1298ec

Contents

.current_func *Get the last function created*

Description

Get the last function created (either via hlo_func or local_func), which is not returned yet.

Usage

```
.current_func()
```

Value

A Func object.

.current_module *Get the current module*

Description

Get the current module created via hlo_module or local_module.

Usage

```
.current_module()
```

Value

A Module object.

BoolAttr *BoolAttr*

Description

An attribute holding a boolean value.

Usage

```
BoolAttr(name, value)
```

Arguments

name	(character(1)) The name of the attribute.
value	(logical(1)) The boolean value.

Value

BoolAttr

Constant *Constant*

Description

This represents a constant value.

Usage

```
Constant(data, type)
```

Arguments

data	(any) The value of the constant.
type	(TensorType) The type of the constant.

Value

Constant

---

constant_attr *Create a ConstantAttr from R values*

Description

Helper function to create a ConstantAttr from R values.

Usage

```
constant_attr(name, value, dtype = NULL, shape = NULL, simplify_dense = TRUE)
```

Arguments

name	(character(1)) The name of the attribute.
value	(any) The R value to convert to a constant.
dtype	(character(1) | NULL) The dtype of the constant. If NULL, inferred from value.
shape	(integer() | NULL) The shape of the constant. If NULL, inferred from value.
simplify_dense	(logical(1)) Whether to simplify dense representation. Set to FALSE for multi-dimensional arrays.

Value

(ConstantAttr)

ConstantAttr *ConstantAttr*

Description

An attribute holding a constant value.

Usage

```
ConstantAttr(name, value, simplify_dense = TRUE)
```

Arguments

name	(character(1)) The name of the attribute.
value	(Constant) The value of the attribute.
simplify_dense	(logical(1)) Whether to simplify dense representation. Set to FALSE for multi-dimensional arrays.

Value

(ConstantAttr)

CustomOpBackendConfig *CustomOpBackendConfig*

Description

A backend configuration as a list of typed attributes for custom operations. Each element must be a `BoolAttr`, `StringAttr`, or `ScalarAttr` for now. All attribute names must be unique.

Usage

```
CustomOpBackendConfig(items = list())
```

Arguments

`items`	(`list`) A list of `BoolAttr`, `StringAttr`, or `ScalarAttr` objects.

Value

`CustomOpBackendConfig`

DotDimensionNumbers *DotDimensionNumbers*

Description

Represents the dot dimension numbers.

Usage

```
DotDimensionNumbers(contracting_dims, batching_dims = NULL)
```

Arguments

`contracting_dims`	(`integer()`) The contracting dimensions.
`batching_dims`	(`integer()` | `NULL`) The batching dimensions.

error_concatenate_shapes *ErrorConcatenateShapes*

Description

Error when input shapes don't match (except at given dimensions)

Usage

```
error_concatenate_shapes(
  dimensions,
  shapes,
  call = sys.call(-1)[1L],
  class = character(),
  signal = TRUE
)
```

Arguments

dimensions	(integer()) The dimensions where shapes may differ (0-based)
shapes	(list() of Shape) The input shapes
call	(call or NULL) Call that generated the error
class	(character()) Additional classes to prepend
signal	(logical(1)) Whether to signal the error (default TRUE)

error_dim_size_mismatch *ErrorDimSizeMismatch*

Description

Error when a dimension size doesn't match the expected size at a given index

Usage

```
error_dim_size_mismatch(
  arg1,
  arg2,
  dim1,
  dim2,
  shape1,
  shape2,
  call = sys.call(-1)[1L],
  class = character(),
  signal = TRUE
)
```

Arguments

arg1	(character(1)) Name of the first argument (e.g. "operand")
arg2	(character(1)) Name of the second argument (e.g. "result")
dim1	(integer(1)) Dimension index in arg1 (0-based)
dim2	(integer(1)) Dimension index in arg2 (0-based)
shape1	(integer()) Complete shape of arg1
shape2	(integer()) Complete shape of arg2
call	(call or NULL) Call that generated the error
class	(character()) Additional classes to prepend
signal	(logical(1)) Whether to signal the error (default TRUE)

---

error_dimension_uniqueness

ErrorDimensionUniqueness

Description

Error when dimension indices are not unique

Usage

```
error_dimension_uniqueness(
  arg,
  dimensions,
  call = sys.call(-1)[1L],
  class = character(),
  signal = TRUE
)
```

Arguments

arg	(character(1)) Name of the argument that caused the error
dimensions	(integer()) The dimension indices that are not unique.
call	(call or NULL) Call that generated the error
class	(character()) Additional classes to prepend
signal	(logical(1)) Whether to signal the error (default TRUE)

---

error_index_in_set *ErrorIndexInSet*

Description

Error when an index is found in a forbidden set

Usage

```
error_index_in_set(
  arg1,
  arg2,
  index,
  set,
  call = sys.call(-1)[1L],
  class = character(),
  signal = TRUE
)
```

Arguments

arg1	(character(1)) Name of the argument containing the index
arg2	(character(1)) Name of the argument containing the set
index	(integer(1)) The index that was found in the set
set	(integer()) The set that should not contain the index
call	(call or NULL) Call that generated the error
class	(character()) Additional classes to prepend
signal	(logical(1)) Whether to signal the error (default TRUE)

---

error_index_out_of_bounds

ErrorIndexOutOfBounds

Description

Error when an index is outside the valid range [lower, upper)

Usage

```
error_index_out_of_bounds(
  arg,
  index,
  lower,
  upper,
  call = sys.call(-1),
  class = character(),
  signal = TRUE
)
```

Arguments

arg	(character(1)) Name of the argument that caused the error
index	(integer()) The observed index value(s).
lower	(integer(1)) Lower bound of valid range.

upper (integer(1))
Upper bound of valid range, exclusive.

call (call or NULL)
Call that generated the error

class (character())
Additional classes to prepend

signal (logical(1))
Whether to signal the error (default TRUE)

---

error_indices_not_sorted *ErrorIndicesNotSorted*

Description

Error when indices are not sorted in ascending order

Usage

```
error_indices_not_sorted(
  arg,
  indices,
  call = sys.call(-1)[1L],
  class = character(),
  signal = TRUE
)
```

Arguments

arg (character(1))
Name of the argument that caused the error

indices (integer())
The indices that are not sorted.

call (call or NULL)
Call that generated the error

class (character())
Additional classes to prepend

signal (logical(1))
Whether to signal the error (default TRUE)

error_permute_index *ErrorPermuteIndex*

Description

Error when permutation values are invalid (not a valid permutation of indices)

Usage

```
error_permute_index(
  arg,
  permutation,
  expected,
  call = sys.call(-1)[1L],
  class = character(),
  signal = TRUE
)
```

Arguments

arg	(character(1)) Name of the argument that caused the error
permutation	(integer()) The permutation values that are invalid.
expected	(integer()) The expected indices to be permuted.
call	(call or NULL) Call that generated the error
class	(character()) Additional classes to prepend
signal	(logical(1)) Whether to signal the error (default TRUE)

error_stablehlo *ErrorStablehlo*

Description

Base error class for all stablehlo errors

Usage

```
error_stablehlo(
  call = sys.call(-1)[1L],
  ...,
  class = character(),
  signal = TRUE
)
```

Arguments

call	(call or NULL) Call that generated the error
...	Additional fields to store in the condition
class	(character()) Additional classes to prepend
signal	(logical(1)) Whether to signal the error (default TRUE)

---

error_unequal_types *ErrorUnequalTypes*

Description

Error when types at the same index in two lists don't match

Usage

```
error_unequal_types(
  arg1,
  arg2,
  index,
  expected,
  actual1,
  actual2,
  call = sys.call(-1)[1L],
  class = character(),
  signal = TRUE
)
```

Arguments

arg1	(character(1)) Name of the first argument
arg2	(character(1)) Name of the second argument

index	(integer(1)) The index where types don't match (0-based)
expected	(character(1)) Description of what was expected
actual1	Type from the first argument (any object with a cli_format method)
actual2	Type from the second argument (any object with a cli_format method)
call	(call or NULL) Call that generated the error
class	(character()) Additional classes to prepend
signal	(logical(1)) Whether to signal the error (default TRUE)

`error_unexpected_list_type` *ErrorUnexpectedListType*

Description

Error when an element in a list has an unexpected type

Usage

```
error_unexpected_list_type(
  arg,
  index,
  expected,
  actual,
  call = sys.call(-1)[1L],
  class = character(),
  signal = TRUE
)
```

Arguments

arg	(character(1)) Name of the argument
index	(integer(1)) The index where the type is unexpected (0-based)
expected	(character(1)) Description of what was expected
actual	(character(1)) What was observed

call	(call or NULL) Call that generated the error
class	(character()) Additional classes to prepend
signal	(logical(1)) Whether to signal the error (default TRUE)

---

format_double *Format Double Array with Scientific Notation*

Description

Formats a double array using scientific notation with 16 digits precision, similar to `formatC(x, digits = 16, format = "e")`.

This is used to embed floating point constants into stableHLO programs.

Usage

```
format_double(x, precision = 64)
```

Arguments

x	(double()) Vector to format.
precision	(integer(1)) Currently supports 32 and 64 bit precisions.

Value

`character()`

Examples

```
format_double(1.23, 32)
format_double(1.23, 64)
```

`Func` *Func*

Description

This represents a function. Note: Func uses reference semantics - modifications to a Func object modify the original.

Usage

```
Func(
  id = FuncId(),
  inputs = FuncInputs(),
  outputs = FuncOutputs(),
  body = FuncBody()
)
```

Arguments

`id`	(`FuncId` The id of the function.
`inputs`	(`FuncInputs` The inputs of the function.
`outputs`	(`FuncOutputs` The outputs of the function.
`body`	(`FuncBody` The body of the function.

Value

A `Func` object.

`FuncBody` *FuncBody*

Description

The body of a `Func`, containing a list of operations.

Usage

```
FuncBody(items = list())
```

Arguments

items	(list() of Op) The operations in the function body.

Value

(FuncBody)

FuncId *FuncId*

Description

This represents the id of a function.

Usage

```
FuncId(id = "main")
```

Arguments

id	The id of the function.

FuncInput *FuncInput*

Description

This represents an input of a Func.

Usage

```
FuncInput(id, type, alias = NULL)
```

Arguments

id	(ValueId) The id of the input.
type	(ValueType) The type of the input.
alias	(integer(1) | NULL) With which output buffer to alias this input.

Value

(FuncInput)

FuncInputs *FuncInputs*

Description

List of FuncInputs.

Usage

```
FuncInputs(items = list())
```

Arguments

items	(list() of FuncInput) The inputs of the function.

Value

(FuncInputs)

FuncOutput *FuncOutput*

Description

This represents an output of a Func.

Usage

```
FuncOutput(type)
```

Arguments

type	(ValueType) The type of the output.

Value

(FuncOutput)

FuncOutputs *FuncOutputs*

Description

List of FuncOutputs.

Usage

```
FuncOutputs(items = list())
```

Arguments

items	(list() of FuncOutput) The outputs of the function.

Value

(FuncOutputs)

FuncValue *FuncValue*

Description

This represents a variable within a function.

Usage

```
FuncValue(value_id, value_type, func)
```

Arguments

value_id	The name of the variable.
value_type	The type of the variable.
func	The function the variable belongs to.

GatherDimensionNumbers *GatherDimensionNumbers*

Description

Represents the gather dimension numbers.

Usage

```
GatherDimensionNumbers(
  offset_dims,
  collapsed_slice_dims,
  operand_batching_dims = integer(),
  start_indices_batching_dims = integer(),
  start_index_map,
  index_vector_dim
)
```

Arguments

offset_dims	(integer()) The offset dimensions.
collapsed_slice_dims	(integer()) The collapsed slice dimensions.
operand_batching_dims	(integer()) The operand batching dimensions.
start_indices_batching_dims	(integer()) The start indices batching dimensions.
start_index_map	(integer()) Maps start indices to operand dimensions.
index_vector_dim	(integer(1)) The index vector dimension.

hlo_cholesky *Cholesky Operator*

Description

See https://openxla.org/stablehlo/spec#cholesky for details.

Usage

```
hlo_cholesky(operand, lower)
```

Arguments

operand, lower (FuncValue)

Details

The values of the other half of the matrix are not guaranteed and backend dependent.

Value

FuncValue

hlo_closure *Create a Closure*

Description

Creates a new function without any arguments that captures the provided variables.

Usage

```
hlo_closure(...)
```

Arguments

...	(FuncValue) The variables to capture.

Value

(list() of FuncValue)

Examples

```
func <- local_func()
x <- hlo_input("x", "f32", shape = c(2, 2))
y <- hlo_input("y", "f32", shape = c(2, 2))
f <- hlo_closure(x, y)
print(f)
```

`hlo_constant` *Create a Constant*

Description

Create either a "scalar" (`hlo_scalar`) or tensor (`hlo_tensor`) constant. Strictly speaking, stableHLO "scalars" are simply tensors with 0 dimensions. To create an empty constant (at least one dimension is 0), use `hlo_empty`.

Usage

```
hlo_scalar(value, ..., dtype = NULL, func = NULL)

hlo_tensor(value, ..., dtype = NULL, shape = NULL, func = NULL)

hlo_empty(dtype, shape, func = NULL)

infer_types_constant(value)
```

Arguments

`value`	(any) Value from which to create a constant.
`...`	(any) Additional arguments.
`dtype`	(`character(1)`) One of: bool, i8, i16, i32, i64, ui8, ui16, ui32, ui64, f32, f64.
`func`	(`Func`) The function to add the constant to. Per default, uses the last function created with `hlo_func` or `local_func`.
`shape`	(`integer()`) The shape

Examples

```
hlo_scalar(1L, dtype = "i32", func = Func())
hlo_scalar(1, dtype = "f32", func = Func())
hlo_scalar(TRUE, func = Func())
hlo_tensor(array(c(1, 2, 3, 4), dim = c(1, 4)), dtype = "f32", func = Func())
hlo_empty(dtype = "f32", shape = c(0, 3), func = Func())
```

`hlo_custom_call` *Custom Call Operation*

Description

Create a custom call operation that invokes an external function via the FFI (Foreign Function Interface) API.

Note that the attributes `called_computations` and `output_operand_aliases` are not implemented yet.

Usage

```
hlo_custom_call(
  ...,
  call_target_name,
  api_version = 4L,
  has_side_effect,
  backend_config = NULL,
  output_types = NULL,
  operand_layouts = NULL,
  result_layouts = NULL
)
```

Arguments

`...`	(FuncValue) Input values to pass to the custom call.
`call_target_name`	(`character(1)`) The name of the registered custom function to call.
`api_version`	(`integer(1)`) The API version. Default is 4.
`has_side_effect`	(`logical(1)`) Whether the custom call has side effects.
`backend_config`	(CustomOpBackendConfig | `NULL`) Optional backend configuration.
`output_types`	(`list` of ValueType | `NULL`) The output types of the custom call. Default is NULL (no outputs).
`operand_layouts`	(`list` of `integer()` | `NULL`) Layouts for each operand in minor-to-major order. Each element is an integer vector specifying the dimension order. For example, `c(0L, 1L)` means column-major (dimension 0 varies fastest), while `c(1L, 0L)` means row-major. Default `NULL` means no layout constraint.
`result_layouts`	(`list` of `integer()` | `NULL`) Layouts for each result in minor-to-major order. Same format as `operand_layouts`.

Value

(FuncValue | list() | NULL)
The output value(s), or NULL for side-effect only calls.

---

hlo_func *Create a function*

Description

Both functions create a new Func with the given id which is afterwards accessible via .current_func(). Functions receiving a Func as an argument (such as hlo_input, hlo_add, ...) usually use .current_func() by default. You can also directly create a function using Func(), which will *not* be accessible this way.

Differences between the two functions:

- local_func removes the function when exiting the current scope, whereas hlo_func does not.
- hlo_func discards the previously built function(s), whereas local_func does not: after a function created by local_func is either cleaned up automatically (by exiting the scope) or the function is finalized via hlo_return, the previously built function is restored, i.e., accessible via .current_func(). To build nested functions (e.g. to create a closure that is passed to another op), use local_func instead of hlo_func.

Usage

```
hlo_func(id = "main")

local_func(id = "main", envir = parent.frame())
```

Arguments

id	(character(1) The id of the function.
envir	(environment) Environment where exit handler will be registered for cleaning up the Func if it was not returned yet.

Value

A Func object.

hlo_input *Create a input to a function*

Description

Create a input to a function

Usage

```
hlo_input(name, dtype, shape = integer(), func = .current_func(), alias = NULL)
```

Arguments

name	(character(1)) The name of the parameter.
dtype	(ValueType) The data type of the parameter. Can contain digits, letters and underscores. If it starts with a digit, it can only contain digits. Otherwise it must start with a letter.
shape	(integer()) The shape of the parameter. Use integer() for scalars.
func	(Func) The function id of the parameter. Per default, uses the last function created with hlo_func.
alias	(integer(1) or NULL) If integer, marks this input as alias with the given output index (0-based).

Examples

```
func <- hlo_func()
x <- hlo_input("x", "f32", shape = c(2, 2))
print(x)

# You can combine multiple inputs as follows:
c(
  hlo_input("x", "f32", shape = c(2, 2)),
  hlo_input("y", "f32", shape = c(2, 2))
)
```

hlo_module *Create a module*

Description

Both functions create a new `Module` which is afterwards accessible via `.current_module()`. Functions created with `hlo_func` or `local_func` will automatically register into the current module. The module is finalized when a function named `"main"` is returned via `hlo_return`.

Differences between the two functions:

- `local_module` removes the module when exiting the current scope, whereas `hlo_module` does not.
- `hlo_module` discards the previously built module(s), whereas `local_module` does not.

Usage

```
hlo_module()

local_module(envir = parent.frame())
```

Arguments

`envir`	(`environment`) Environment where exit handler will be registered for cleaning up the `Module` if it was not finalized yet.

Value

A `Module` object.

index_vec *index_vec*

Description

Wraps an integer vector marking it as containing 0-based index values.

Usage

```
index_vec(x)
```

Arguments

`x`	(`integer()`) Integer vector of indices.

Value

An integer vector with additional class "IndexVector".

---

infer_types_abs *Abs Operator*

Description

See https://openxla.org/stablehlo/spec#abs for details.

Usage

```
infer_types_abs(operand)

hlo_abs(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_acos *Acos Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chloacos_chloacosop for details.

Usage

```
infer_types_acos(operand)

hlo_acos(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

`infer_types_acosh` *Acosh Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chloacosh_chloacoshop for details.

Usage

```
infer_types_acosh(operand)

hlo_acosh(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

`infer_types_add` *Add Operator*

Description

See https://openxla.org/stablehlo/spec#add for details.

Usage

```
infer_types_add(lhs, rhs)

hlo_add(lhs, rhs)
```

Arguments

lhs, rhs	(FuncValue)

Value

FuncValue

---

infer_types_after_all *AfterAll Operator*

Description

See https://openxla.org/stablehlo/spec#after_all for details.

Usage

```
infer_types_after_all(...)

hlo_after_all(...)
```

Arguments

...	(FuncValue)

Value

FuncValue

---

infer_types_and *And Operator*

Description

See https://openxla.org/stablehlo/spec#and for details.

Usage

```
infer_types_and(lhs, rhs)

hlo_and(lhs, rhs)
```

Arguments

lhs, rhs	(FuncValue)

Value

FuncValue

`infer_types_asin` *Asin Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chloasin_chloasinop for details.

Usage

```
infer_types_asin(operand)

hlo_asin(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

`infer_types_asinh` *Asinh Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chloasinh_chloasinhop for details.

Usage

```
infer_types_asinh(operand)

hlo_asinh(operand)
```

Arguments

operand (FuncValue)

Value

FuncValue

---

infer_types_atan *Atan Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chloatan_chloatanop for details.

Usage

```
infer_types_atan(operand)

hlo_atan(operand)
```

Arguments

operand (FuncValue)

Value

FuncValue

---

infer_types_atan2 *Atan2 Operator*

Description

See https://openxla.org/stablehlo/spec#atan2 for details.

Usage

```
infer_types_atan2(lhs, rhs)

hlo_atan2(lhs, rhs)
```

Arguments

lhs, rhs (FuncValue)

Value

FuncValue

---

infer_types_atanh *Atanh Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chloatanh_chloatanhop for details.

Usage

```
infer_types_atanh(operand)

hlo_atanh(operand)
```

Arguments

operand (FuncValue)

Value

FuncValue

---

infer_types_bessel_i1e *BesselI1e Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlobessel_i1e_chlobessel_i1eop for details.

Usage

```
infer_types_bessel_i1e(operand)

hlo_bessel_i1e(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_bitcast_convert

BitcastConvert Operator

Description

See https://openxla.org/stablehlo/spec#bitcast_convert for details.

Usage

```
infer_types_bitcast_convert(operand, dtype)

hlo_bitcast_convert(operand, dtype)
```

Arguments

operand, dtype	(FuncValue)

Value

FuncValue

infer_types_broadcast_in_dim *BroadcastInDim Operator*

Description

See https://openxla.org/stablehlo/spec#broadcast_in_dim for details.

Usage

```
infer_types_broadcast_in_dim(operand, broadcast_dimensions, shape)

hlo_broadcast_in_dim(operand, broadcast_dimensions, shape)
```

Arguments

operand, broadcast_dimensions, shape
(FuncValue)

Value

FuncValue

infer_types_call *Call a Function*

Description

Calls a named function from within the current function being built. The callee must already be finalized via hlo_return.

Usage

```
infer_types_call(callee, ...)

hlo_call(callee, ..., simplify = TRUE)
```

Arguments

callee	(Func) The function to call. Must be a finalized Func.
...	(FuncValue) The arguments to pass to the callee.
simplify	(logical(1)) If TRUE (default) and the callee has a single output, return a single FuncValue instead of a list.

Value

FuncValue or list() of FuncValues.

---

infer_types_case *Case Operator*

Description

See https://openxla.org/stablehlo/spec#case for details.

Usage

```
infer_types_case(index, ...)

hlo_case(index, ...)
```

Arguments

index, ...	(FuncValue)

Value

FuncValue

---

infer_types_cbrt *Cbrt Operator*

Description

See https://openxla.org/stablehlo/spec#cbrt for details.

Usage

```
infer_types_cbrt(operand)

hlo_cbrt(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_ceil *Ceil Operator*

Description

See https://openxla.org/stablehlo/spec#ceil for details.

Usage

```
infer_types_ceil(operand)

hlo_ceil(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_clamp *Clamp Operator*

Description

See https://openxla.org/stablehlo/spec#clamp for details.

Usage

```
infer_types_clamp(min, operand, max)

hlo_clamp(min, operand, max)
```

Arguments

min, operand, max	(FuncValue)

Value

FuncValue

infer_types_compare *Compare Operator*

Description

See https://openxla.org/stablehlo/spec#compare for details.

Usage

```
infer_types_compare(lhs, rhs, comparison_direction, compare_type)

hlo_compare(lhs, rhs, comparison_direction, compare_type)
```

Arguments

lhs, rhs, comparison_direction, compare_type
(FuncValue)

Value

FuncValue

infer_types_concatenate *Concatenate Operator*

Description

See https://openxla.org/stablehlo/spec#concatenate for details.

Usage

```
infer_types_concatenate(..., dimension)

hlo_concatenate(..., dimension)
```

Arguments

..., dimension (FuncValue)

Value

FuncValue

infer_types_convert *Convert Operator*

Description

See https://openxla.org/stablehlo/spec#convert for details.

Usage

```
infer_types_convert(operand, dtype)

hlo_convert(operand, dtype)
```

Arguments

operand, dtype (FuncValue)

Value

FuncValue

infer_types_cosh *Cosh Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlocosh_chlocoshop for details.

Usage

```
infer_types_cosh(operand)

hlo_cosh(operand)
```

Arguments

operand (FuncValue)

Value

FuncValue

infer_types_cosine *Cosine Operator*

Description

See https://openxla.org/stablehlo/spec#cosine for details.

Usage

```
infer_types_cosine(operand)

hlo_cosine(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_count_leading_zeros *CountLeadingZeros Operator*

Description

See https://openxla.org/stablehlo/spec#count_leading_zeros for details.

Usage

```
infer_types_count_leading_zeros(operand)

hlo_count_leading_zeros(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

`infer_types_custom_call` *Infer types for custom call*

Description

Infer the output types for a custom call operation.

Usage

```
infer_types_custom_call(
  ...,
  call_target_name,
  api_version,
  has_side_effect,
  backend_config,
  output_types,
  operand_layouts,
  result_layouts
)
```

Arguments

`...`	Input values.
`call_target_name`	(`character(1)`) The name of the custom function to call.
`api_version`	(`integer(1)`) The API version.
`has_side_effect`	(`logical(1)`) Whether the custom call has side effects.
`backend_config`	(`list` | `NULL`) Optional backend configuration as a named list.
`output_types`	(`list` of `ValueType` | `NULL`) The output types of the custom call. Default is NULL (no outputs).
`operand_layouts`, `result_layouts`	Layouts (not used for type inference).

Value

(`ValueTypes`)
The output types (empty for side-effect only calls).

infer_types_digamma *Digamma Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlodigamma_chlodigammaop for details.

Usage

```
infer_types_digamma(operand)

hlo_digamma(operand)
```

Arguments

operand		(FuncValue)

Value

FuncValue

infer_types_divide *Divide Operator*

Description

See https://openxla.org/stablehlo/spec#divide for details.

Usage

```
infer_types_divide(lhs, rhs)

hlo_divide(lhs, rhs)
```

Arguments

lhs, rhs		(FuncValue)

Value

FuncValue

`infer_types_dot_general` *DotGeneral Operator*

Description

See https://openxla.org/stablehlo/spec#dot_general for details.

Usage

```
infer_types_dot_general(lhs, rhs, dot_dimension_numbers)

hlo_dot_general(lhs, rhs, contracting_dims, batching_dims = NULL)
```

Arguments

`lhs`, `rhs`, `contracting_dims`, `batching_dims`
(FuncValue)

`dot_dimension_numbers`
(DotDimensionNumbers)
The dot dimension number.

Value

FuncValue

`infer_types_dynamic_slice` *DynamicSlice Operator*

Description

See https://openxla.org/stablehlo/spec#dynamic_slice for details.

Usage

```
infer_types_dynamic_slice(operand, ..., slice_sizes)

hlo_dynamic_slice(operand, ..., slice_sizes)
```

Arguments

`operand`, `...`, `slice_sizes`
(FuncValue)

Value

FuncValue

`infer_types_dynamic_update_slice` *DynamicUpdateSlice Operator*

Description

See https://openxla.org/stablehlo/spec#dynamic_update_slice for details.

Usage

```
infer_types_dynamic_update_slice(operand, update, ...)

hlo_dynamic_update_slice(operand, update, ...)
```

Arguments

operand, update, ... (FuncValue)

Value

FuncValue

`infer_types_erf` *Erf Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chloerf_chloerfop for details.

Usage

```
infer_types_erf(operand)

hlo_erf(operand)
```

Arguments

operand (FuncValue)

Value

FuncValue

---

infer_types_erf_inv *ErfInv Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chloerf_inv_chloerf_invop for details.

Usage

```
infer_types_erf_inv(operand)

hlo_erf_inv(operand)
```

Arguments

operand (FuncValue)

Value

FuncValue

---

infer_types_erfc *Erfc Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chloerfc_chloerfcop for details.

Usage

```
infer_types_erfc(operand)

hlo_erfc(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_exponential
Exponential Operator

Description

See https://openxla.org/stablehlo/spec#exponential for details.

Usage

```
infer_types_exponential(operand)

hlo_exponential(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_exponential_minus_one *ExponentialMinusOne Operator*

Description

See https://openxla.org/stablehlo/spec#exponential_minus_one for details.

Usage

```
infer_types_exponential_minus_one(operand)

hlo_exponential_minus_one(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_float_biv *Infer types for float binary operations*

Description

Infer the types for float binary operations.

Usage

```
infer_types_float_biv(lhs, rhs)
```

Arguments

lhs	(ValueType) The left-hand side operand.
rhs	(ValueType) The right-hand side operand.

Value

(ValueType)
The inferred type.

infer_types_float_uni *Infer types for float unary operations*

Description

Infer the types for float unary operations.

Usage

```
infer_types_float_uni(operand)
```

Arguments

operand	(ValueType) The operand.

Value

(ValueType)
The inferred type.

infer_types_floor *Floor Operator*

Description

See https://openxla.org/stablehlo/spec#floor for details.

Usage

```
infer_types_floor(operand)

hlo_floor(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_gather *Gather Operator*

Description

See https://openxla.org/stablehlo/spec#gather for details.

Usage

```
infer_types_gather(
  operand,
  start_indices,
  gather_dimension_numbers,
  slice_sizes,
  indices_are_sorted
)

hlo_gather(
  operand,
  start_indices,
  gather_dimension_numbers,
  slice_sizes,
  indices_are_sorted = FALSE
)
```

Arguments

operand, start_indices
: (FuncValue)

gather_dimension_numbers
: (GatherDimensionNumbers)
The gather dimension numbers.

slice_sizes
: (integer())
The sizes of the slices to gather.

indices_are_sorted
: (logical(1))
Whether indices are sorted.

Value

FuncValue

infer_types_generic_biv

Infer types for binary operations

Description

Infer the types for binary operations.

Usage

```
infer_types_generic_biv(lhs, rhs)
```

Arguments

lhs	(ValueType) The left-hand side operand.
rhs	(ValueType) The right-hand side operand.

Value

(ValueType)
The inferred type.

infer_types_generic_uni

Infer types for unary operations

Description

Infer the types for unary operations.

Usage

```
infer_types_generic_uni(operand)
```

Arguments

operand	(ValueType) The operand.

Value

(ValueType)
The inferred type.

infer_types_if *If Operator*

Description

See https://openxla.org/stablehlo/spec#if for details.

Usage

```
infer_types_if(pred, true_branch, false_branch)

hlo_if(pred, true_branch, false_branch, simplify = TRUE)
```

Arguments

pred, true_branch, false_branch	(FuncValue)
simplify	(logical(1)) Whether to simplify results by unpacking lists of length 1 into their single element.

Value

FuncValue

infer_types_integer_uni *Infer types for integer unary operations*

Description

Infer the types for integer unary operations.

Usage

```
infer_types_integer_uni(operand)
```

Arguments

operand	(ValueType) The operand.

Value

(ValueType)
The inferred type.

---

infer_types_integerish_biv

Infer types for boolean integerish operations

Description

Infer the types for integerish (bool or int) binary operations.

Usage

```
infer_types_integerish_biv(lhs, rhs)
```

Arguments

lhs	(ValueType) The left-hand side operand.
rhs	(ValueType) The right-hand side operand.

Value

(ValueType)
The inferred type.

---

infer_types_integerish_uni

Infer types for integerish unary operations

Description

Infer the types for integerish (bool or int) unary operations.

Usage

```
infer_types_integerish_uni(operand)
```

Arguments

operand	(ValueType) The operand.

Value

(ValueType)
The inferred type.

`infer_types_iota` *Iota Operator*

Description

See https://openxla.org/stablehlo/spec#iota for details.

Usage

```
infer_types_iota(iota_dimension, dtype, shape)

hlo_iota(iota_dimension, dtype, shape, func = NULL)
```

Arguments

`iota_dimension`	(integer(1)) The dimension along which to generate increasing values. Must be in range [0, rank(output)).
`dtype`	(character(1)) The data type of the output tensor. One of: bool, i8, i16, i32, i64, ui8, ui16, ui32, ui64, f32, f64 (excluding boolean).
`shape`	(integer()) The shape of the output tensor.
`func`	(Func) The function to add the operation to. Per default, uses the last function created with hlo_func or local_func.

Value

FuncValue

infer_types_is_finite *IsFinite Operator*

Description

See https://openxla.org/stablehlo/spec#is_finite for details.

Usage

```
infer_types_is_finite(operand)

hlo_is_finite(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_is_inf *IsInf Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlois_inf_chlois_infop for details.

Usage

```
infer_types_is_inf(operand)

hlo_is_inf(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_is_neg_inf

IsNegInf Operator (CHLO)

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlois_neg_inf_chlois_neg_infop for details.

Usage

```
infer_types_is_neg_inf(operand)

hlo_is_neg_inf(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_is_pos_inf

IsPosInf Operator (CHLO)

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlois_pos_inf_chlois_pos_infop for details.

Usage

```
infer_types_is_pos_inf(operand)

hlo_is_pos_inf(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_lgamma *Lgamma Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlolgamma_chlolgammaop for details.

Usage

```
infer_types_lgamma(operand)

hlo_lgamma(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_log *Log Operator*

Description

See https://openxla.org/stablehlo/spec#log for details.

Usage

```
infer_types_log(operand)

hlo_log(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_log_plus_one *LogPlusOne Operator*

Description

See https://openxla.org/stablehlo/spec#log_plus_one for details.

Usage

```
infer_types_log_plus_one(operand)

hlo_log_plus_one(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_logistic *Logistic Operator*

Description

See https://openxla.org/stablehlo/spec#logistic for details.

Usage

```
infer_types_logistic(operand)

hlo_logistic(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_maximum *Maximum Operator*

Description

See https://openxla.org/stablehlo/spec#maximum for details.

Usage

```
infer_types_maximum(lhs, rhs)

hlo_maximum(lhs, rhs)
```

Arguments

lhs, rhs		(FuncValue)

Value

FuncValue

---

infer_types_minimum *Minimum Operator*

Description

See https://openxla.org/stablehlo/spec#minimum for details.

Usage

```
infer_types_minimum(lhs, rhs)

hlo_minimum(lhs, rhs)
```

Arguments

lhs, rhs		(FuncValue)

Value

FuncValue

---

infer_types_multiply *Multiply Operator*

Description

See https://openxla.org/stablehlo/spec#multiply for details.

Usage

```
infer_types_multiply(lhs, rhs)

hlo_multiply(lhs, rhs)
```

Arguments

lhs, rhs		(FuncValue)

Value

FuncValue

---

infer_types_negate *Negate Operator*

Description

See https://openxla.org/stablehlo/spec#negate for details.

Usage

```
infer_types_negate(operand)

hlo_negate(operand)
```

Arguments

operand		(FuncValue)

Value

FuncValue

`infer_types_not` *Not Operator*

Description

See https://openxla.org/stablehlo/spec#not for details.

Usage

```
infer_types_not(operand)

hlo_not(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

`infer_types_numeric_biv`
Infer types for numeric binary operations

Description

Infer the types for numeric binary operations.

Usage

```
infer_types_numeric_biv(lhs, rhs)
```

Arguments

lhs	(ValueType) The left-hand side operand.
rhs	(ValueType) The right-hand side operand.

Value

(ValueType)
The inferred type.

`infer_types_numeric_uni` *Infer types for numeric unary operations*

Description

Infer the types for numeric unary operations.

Usage

```
infer_types_numeric_uni(operand)
```

Arguments

operand	(ValueType) The operand.

Value

(ValueType)
The inferred type.

`infer_types_or` *Or Operator*

Description

See https://openxla.org/stablehlo/spec#or for details.

Usage

```
infer_types_or(lhs, rhs)

hlo_or(lhs, rhs)
```

Arguments

lhs, rhs	(FuncValue)

Value

FuncValue

infer_types_pad *Pad Operator*

Description

See https://openxla.org/stablehlo/spec#pad for details.

Usage

```
infer_types_pad(
  operand,
  padding_value,
  edge_padding_low,
  edge_padding_high,
  interior_padding
)

hlo_pad(
  operand,
  padding_value,
  edge_padding_low,
  edge_padding_high,
  interior_padding
)
```

Arguments

operand, padding_value, edge_padding_low, edge_padding_high, interior_padding
(FuncValue)

Value

FuncValue

infer_types_polygamma *Polygamma Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlopolygamma_chlopolygammaop for details.

Usage

```
infer_types_polygamma(n, x)

hlo_polygamma(n, x)
```

Arguments

n, x	(FuncValue)

Value

FuncValue

---

infer_types_popcnt *Popcnt Operator*

Description

See https://openxla.org/stablehlo/spec#popcnt for details.

Usage

```
infer_types_popcnt(operand)

hlo_popcnt(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_power *Power Operator*

Description

See https://openxla.org/stablehlo/spec#power for details.

Usage

```
infer_types_power(lhs, rhs)

hlo_power(lhs, rhs)
```

Arguments

lhs, rhs	(FuncValue)

Value

FuncValue

infer_types_reduce *Reduce Operator*

Description

See https://openxla.org/stablehlo/spec#reduce for details.

Usage

```
infer_types_reduce(inputs, init_values, body, dimensions)

hlo_reduce(inputs, init_values, dimensions, body)
```

Arguments

inputs, init_values, dimensions, body	(FuncValue)

Value

FuncValue

infer_types_reduce_window *ReduceWindow Operator*

Description

See https://openxla.org/stablehlo/spec#reduce_window for details.

Usage

```
infer_types_reduce_window(
  ...,
  body,
  window_dimensions,
  window_strides,
  base_dilations,
  window_dilations,
  padding
)

hlo_reduce_window(
  inputs,
  init_values,
  window_dimensions,
  window_strides,
  base_dilations,
  window_dilations,
  padding,
  body
)
```

Arguments

`...`	(Inputs, Init values)
`body`	(Func) The reduction function to apply to each window.
`window_dimensions`	(integer()) The size of the window in each dimension.
`window_strides`	(integer()) The stride of the window in each dimension.
`base_dilations`	(integer()) The dilation factor for the input tensor.
`window_dilations`	(integer()) The dilation factor for the window.

padding (matrix)
A matrix with shape [rank, 2] specifying the padding before and after each dimension.

inputs (list() of FuncValue)
The input tensor(s) to apply the reduction to.

init_values (list() of FuncValue)
The initial value(s) for the reduction. Must be 0-D tensors.

---

infer_types_remainder *Remainder Operator*

Description

See https://openxla.org/stablehlo/spec#remainder for details.

Usage

```
infer_types_remainder(lhs, rhs)

hlo_remainder(lhs, rhs)
```

Arguments

lhs, rhs (FuncValue)

Value

FuncValue

---

infer_types_reshape *Reshape Operator*

Description

See https://openxla.org/stablehlo/spec#reshape for details.

Usage

```
infer_types_reshape(operand, shape)

hlo_reshape(operand, shape)
```

Arguments

operand, shape (FuncValue)

Value

FuncValue

---

infer_types_return *Return Values*

Description

Specifies the return values of a Func and finalize it.

Usage

```
infer_types_return(...)

hlo_return(..., func = .current_func())
```

Arguments

...	(FuncValue) Return values. There must be at least one.
func	(Func) The function. Default is to use .current_func().

Value

(Func)

---

infer_types_reverse *Reverse Operator*

Description

See https://openxla.org/stablehlo/spec#reverse for details.

Usage

```
infer_types_reverse(operand, dimensions)

hlo_reverse(operand, dimensions)
```

Arguments

operand, dimensions
(FuncValue)

Value

FuncValue

---

infer_types_rng_bit_generator
RngBitGenerator Operator

Description

See https://openxla.org/stablehlo/spec#rng_bit_generator for details.

Usage

```
infer_types_rng_bit_generator(initial_state, rng_algorithm, dtype, shape)

hlo_rng_bit_generator(
  initial_state,
  rng_algorithm = c("DEFAULT", "THREE_FRY", "PHILOX"),
  dtype,
  shape
)
```

Arguments

initial_state, rng_algorithm, dtype, shape
(FuncValue)

Value

FuncValue

`infer_types_round_nearest_afz` *RoundNearestAfz Operator*

Description

See https://openxla.org/stablehlo/spec#round_nearest_afz for details.

Usage

```
infer_types_round_nearest_afz(operand)

hlo_round_nearest_afz(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

`infer_types_round_nearest_even` *RoundNearestEven Operator*

Description

See https://openxla.org/stablehlo/spec#round_nearest_even for details.

Usage

```
infer_types_round_nearest_even(operand)

hlo_round_nearest_even(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_rsqrt *Rsqrt Operator*

Description

See https://openxla.org/stablehlo/spec#rsqrt for details.

Usage

```
infer_types_rsqrt(operand)

hlo_rsqrt(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

infer_types_scatter *Scatter Operator*

Description

See https://openxla.org/stablehlo/spec#scatter for details.

Usage

```
infer_types_scatter(
  inputs,
  scatter_indices,
  updates,
  scatter_dimension_numbers,
  indices_are_sorted,
  unique_indices,
  update_computation
)

hlo_scatter(
  inputs,
  scatter_indices,
  updates,
```

```
  scatter_dimension_numbers,
  indices_are_sorted = FALSE,
  unique_indices = FALSE,
  update_computation
)
```

Arguments

inputs, scatter_indices, updates
: (FuncValue)

scatter_dimension_numbers
: (ScatterDimensionNumbers)
The scatter dimension numbers.

indices_are_sorted
: (logical(1))
Whether indices are sorted.

unique_indices
: (logical(1))
Whether indices are unique.

update_computation
: (Func)
The update computation function.

Value

FuncValue

---

infer_types_select *Select Operator*

Description

See https://openxla.org/stablehlo/spec#select for details.

Usage

```
infer_types_select(pred, on_true, on_false)

hlo_select(pred, on_true, on_false)
```

Arguments

pred, on_true, on_false
: (FuncValue)

Value

FuncValue

`infer_types_shift_left`
ShiftLeft Operator

Description

See https://openxla.org/stablehlo/spec#shift_left for details.

Usage

```
infer_types_shift_left(lhs, rhs)

hlo_shift_left(lhs, rhs)
```

Arguments

lhs, rhs	(FuncValue)

Value

FuncValue

`infer_types_shift_right_arithmetic`
ShiftRightArithmetic Operator

Description

See https://openxla.org/stablehlo/spec#shift_right_arithmetic for details.

Usage

```
infer_types_shift_right_arithmetic(lhs, rhs)

hlo_shift_right_arithmetic(lhs, rhs)
```

Arguments

lhs, rhs	(FuncValue)

Value

FuncValue

---

infer_types_shift_right_logical *ShiftRightLogical Operator*

Description

See https://openxla.org/stablehlo/spec#shift_right_logical for details.

Usage

```
infer_types_shift_right_logical(lhs, rhs)

hlo_shift_right_logical(lhs, rhs)
```

Arguments

lhs, rhs	(FuncValue)

Value

FuncValue

---

infer_types_sign *Sign Operator*

Description

See https://openxla.org/stablehlo/spec#sign for details.

Usage

```
infer_types_sign(operand)

hlo_sign(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_sine *Sine Operator*

Description

See https://openxla.org/stablehlo/spec#sine for details.

Usage

```
infer_types_sine(operand)

hlo_sine(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_sinh *Sinh Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlosinh_chlosinhop for details.

Usage

```
infer_types_sinh(operand)

hlo_sinh(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

`infer_types_slice` *Slice Operator*

Description

See https://openxla.org/stablehlo/spec#slice for details.

Usage

```
infer_types_slice(operand, start_indices, limit_indices, strides)

hlo_slice(operand, start_indices, limit_indices, strides)
```

Arguments

operand, start_indices, limit_indices, strides
(FuncValue)

Value

FuncValue

`infer_types_sort` *Sort Operator*

Description

See https://openxla.org/stablehlo/spec#sort for details.

Usage

```
infer_types_sort(..., dimension, is_stable, comparator)

hlo_sort(..., dimension, is_stable, comparator)
```

Arguments

..., dimension, is_stable, comparator
(FuncValue)

Value

FuncValue

---

infer_types_sqrt *Sqrt Operator*

Description

See https://openxla.org/stablehlo/spec#sqrt for details.

Usage

```
infer_types_sqrt(operand)

hlo_sqrt(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_square *Square Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlosquare_chlosquareop for details.

Usage

```
infer_types_square(operand)

hlo_square(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_subtract *Subtract Operator*

Description

See https://openxla.org/stablehlo/spec#subtract for details.

Usage

```
infer_types_subtract(lhs, rhs)

hlo_subtract(lhs, rhs)
```

Arguments

lhs, rhs		(FuncValue)

Value

FuncValue

---

infer_types_tan *Tan Operator*

Description

See https://openxla.org/stablehlo/spec#tan for details.

Usage

```
infer_types_tan(operand)

hlo_tan(operand)
```

Arguments

operand		(FuncValue)

Value

FuncValue

---

infer_types_tanh *Tanh Operator*

Description

See https://openxla.org/stablehlo/spec#tanh for details.

Usage

```
infer_types_tanh(operand)

hlo_tanh(operand)
```

Arguments

operand	(FuncValue)

Value

FuncValue

---

infer_types_top_k *TopK Operator (CHLO)*

Description

This op is from the CHLO dialect, a higher-level companion to stableHLO that is lowered to stableHLO during compilation. See https://openxla.org/stablehlo/generated/chlo#chlotop_k_chlotop_kop for details.

Usage

```
infer_types_top_k(operand, k)

hlo_top_k(operand, k)
```

Arguments

operand	(FuncValue) Tensor of integer, unsigned integer, or floating-point type with rank >= 1.
k	(integer(1)) Number of top elements to return along the last dimension. Must satisfy 1 <= k <= dim(operand, -1).

Value

FuncValue

A list() of two FuncValues: the top-k values (same dtype as operand) and their indices into the last dimension (dtype i32). Ties are broken by lower index first.

---

infer_types_transpose *Transpose Operator*

Description

See https://openxla.org/stablehlo/spec#transpose for details.

Usage

```
infer_types_transpose(operand, permutation)

hlo_transpose(operand, permutation)
```

Arguments

operand, permutation	(FuncValue)

Value

FuncValue

`infer_types_triangular_solve` *TriangularSolve Operator*

Description

See https://openxla.org/stablehlo/spec#triangular_solve for details.

Usage

```
infer_types_triangular_solve(
  a,
  b,
  left_side,
  lower,
  unit_diagonal,
  transpose_a
)

hlo_triangular_solve(a, b, left_side, lower, unit_diagonal, transpose_a)
```

Arguments

a, b	(FuncValue)
left_side	(logical(1)) If TRUE, solve op(a) * x = b. If FALSE, solve x * op(a) = b.
lower	(logical(1)) If TRUE, use lower triangle of a. If FALSE, use upper triangle.
unit_diagonal	(logical(1)) If TRUE, assume diagonal elements of a are 1.
transpose_a	(character(1)) One of "NO_TRANSPOSE", "TRANSPOSE", or "ADJOINT".

Value

FuncValue

infer_types_while *While Operator*

Description

See https://openxla.org/stablehlo/spec#while for details.

Usage

```
infer_types_while(..., cond, body)

hlo_while(..., cond, body, simplify = TRUE)
```

Arguments

..., cond, body	(FuncValue)
simplify	(logical(1)) Whether to simplify results by unpacking lists of length 1 into their single element.

Value

FuncValue

infer_types_xor *Xor Operator*

Description

See https://openxla.org/stablehlo/spec#xor for details.

Usage

```
infer_types_xor(lhs, rhs)

hlo_xor(lhs, rhs)
```

Arguments

lhs, rhs	(FuncValue)

Value

FuncValue

Module *Module*

Description

A StableHLO module containing one or more `Func` objects. Modules allow defining multiple named functions, where functions can call each other using `hlo_call`.

Note: Module uses reference semantics like `Func`.

Usage

```
Module(funcs = list())
```

Arguments

`funcs`	(`list()` of `Func`) The functions in the module.

Value

A `Module` object.

Op *Op*

Description

This represents a StableHLO operation.

Usage

```
Op(name, inputs, outputs, signature)
```

Arguments

`name`	(`OpName`) The name of the operation.
`inputs`	(`OpInputs`) The inputs to the operation.
`outputs`	(`OpOutputs`) The outputs of the operation.
`signature`	(`OpSignature`) The signature of the operation.

Value

(`Op`)

`OpInputAttr` *OpInputAttr*

Description

Base class for operation input attributes.

Usage

```
OpInputAttr(name, value, dtype)
```

Arguments

`name`	(`character(1)`) The name of the attribute.
`value`	(any) The value of the attribute.
`dtype`	(`tengen::DataType`) The dtype of the attribute.

Value

(`OpInputAttr`)

`OpInputAttrs` *OpInputAttrs*

Description

List of `OpInputAttr`s.

Usage

```
OpInputAttrs(items = list())
```

Arguments

`items`	(`list()` of `OpInputAttr`) The attributes that can be used as inputs to operations.

Value

(`OpInputAttrs`)

OpInputFunc *OpInputFunc*

Description

This represents a function that can be used as input to an operation.

Usage

```
OpInputFunc(inputs, body)
```

Arguments

inputs	(FuncInputs) The inputs of the function.
body	(FuncBody) The body of the function.

Value

(OpInputFunc)

OpInputFuncs *OpInputFuncs*

Description

List of OpInputFuncs.

Usage

```
OpInputFuncs(items = list())
```

Arguments

items	(list() of OpInputFunc) The functions that can be used as inputs to operations.

Value

(OpInputFuncs)

OpInputs *OpInputs*

Description

This represents all the inputs to an operation, including values, functions, and attributes.

Usage

```
OpInputs(
  values,
  funcs = OpInputFuncs(),
  attrs = OpInputAttrs(),
  custom_attrs = list()
)
```

Arguments

values	(OpInputValues) The values used as inputs.
funcs	(OpInputFuncs) The functions used as inputs.
attrs	(OpInputAttrs) The attributes used as inputs.
custom_attrs	(list) Custom attributes. Use this attributes that require custom formatting.

Value

(OpInputs)

OpInputValue *OpInputValue*

Description

This represents a value that can be used as input to an operation.

Usage

```
OpInputValue(id)
```

Arguments

id	(ValueId) The id of the value.

Value

(OpInputValue)

OpInputValues *OpInputValues*

Description

List of OpInputValues.

Usage

```
OpInputValues(items = list())
```

Arguments

items	(list() of OpInputValue) The values that can be used as inputs to operations.

Value

(OpInputValues)

OpName *OpName*

Description

This represents the name of an operation, containing a mnemonic and a dialect.

Usage

```
OpName(mnemonic, dialect = "stablehlo")
```

Arguments

mnemonic	(character(1)) The mnemonic of the operation.
dialect	(character(1)) The MLIR dialect this op belongs to. Defaults to "stablehlo". Use "chlo" for CHLO ops (which lower to StableHLO during compilation).

Value

(OpName)

OpOutput *OpOutput*

Description

This represents an output of an operation.

Usage

```
OpOutput(id = ValueId())
```

Arguments

id	(ValueId) The id of the output.

Value

(OpOutput)

OpOutputs *OpOutputs*

Description

List of OpOutputs.

Usage

```
OpOutputs(items = list())
```

Arguments

items	(list() of OpOutput) The outputs of an operation.

Value

(OpOutputs)

OpSignature *OpSignature*

Description

This represents the signature of an operation, defining its input and output types.

Usage

```
OpSignature(input_types, output_types)
```

Arguments

input_types	(ValueTypes) The types of the inputs.
output_types	(ValueTypes) The types of the outputs.

Value

(OpSignature)

r_to_constant *Convert R value to Constant*

Description

Convert R value to Constant.

Usage

```
r_to_constant(value, dtype = NULL, shape, ...)
```

Arguments

value	(any) The value to convert.
dtype	(character(1)) The dtype of the constant.
shape	(integer()) The shape of the constant.
...	(any) Additional arguments.

repr *Generate string representation for object*

Description

This function generates a string representation of an object. In this package, this is primarily used to convert a Func to its stableHLO string representation.

Usage

```
repr(x, ...)
```

Arguments

x	The object to generate a string representation of.
...	Additional arguments passed to the method.

Value

character(1)

ScalarAttr *ScalarAttr*

Description

An attribute holding a scalar value with an associated dtype.

Usage

```
ScalarAttr(name, value, dtype)
```

Arguments

name	(character(1)) The name of the attribute.
value	(numeric(1) or logical(1)) The scalar value.
dtype	(tengen::DataType) The dtype of the scalar (e.g., IntegerType(32), FloatType(32), BooleanType()).

Value

ScalarAttr

ScatterDimensionNumbers *ScatterDimensionNumbers*

Description

Represents the scatter dimension numbers.

Usage

```
ScatterDimensionNumbers(
  update_window_dims,
  inserted_window_dims,
  input_batching_dims = integer(),
  scatter_indices_batching_dims = integer(),
  scatter_dims_to_operand_dims,
  index_vector_dim
)
```

Arguments

update_window_dims
: (integer())
The update window dimensions.

inserted_window_dims
: (integer())
The inserted window dimensions.

input_batching_dims
: (integer())
The input batching dimensions.

scatter_indices_batching_dims
: (integer())
The scatter indices batching dimensions.

scatter_dims_to_operand_dims
: (integer())
Maps scatter dimensions to operand dimensions.

index_vector_dim
: (integer(1))
The index vector dimension.

Shape *Shape*

Description

Represents the shape of a tensor.

Usage

```
Shape(dims = integer())
```

Arguments

dims	(integer())

Value

Shape

StringAttr *StringAttr*

Description

An attribute holding a string value.

Usage

```
StringAttr(name, value)
```

Arguments

name	(character(1)) The name of the attribute.
value	(character(1)) The string value.

Value

StringAttr

TensorType *TensorType*

Description

Represents a tensor type with a specific data type and shape.

Usage

```
TensorType(dtype, shape)
```

Arguments

dtype	(tengen::DataType)
shape	(Shape)

Value

TensorType

to_one_based *Convert 0-based indices to 1-based*

Description

Converts all index_vec fields in a condition object from 0-based to 1-based.

Usage

```
to_one_based(x, ...)
```

Arguments

x	Condition object with index_vec fields.
...	Additional arguments (not used).

Value

Condition object with index_vec fields incremented by 1.

`ValueId` *ValueId*

Description

This represents the name of a `ValueType`.

Usage

```
ValueId(id = NULL)
```

Arguments

`id`	(`character(1)` or `environment`) Either a fixed name or an environment. If using an environment (default), the name will be generated automatically when calling `repr()`, i.e. the first value id will be `%0`, the second `%1`, etc..

Value

`(ValueId)`

`ValueType` *ValueType*

Description

This represents the type of a value.

Usage

```
ValueType(type, shape = NULL)
```

Arguments

`type`	The type of the value (TensorType or TokenType).
`shape`	The shape of the value (only used when type is character).

ValueTypes *ValueTypes*

Description

List of ValueTypes.

Usage

```
ValueTypes(items = list())
```

Arguments

items	(list() of ValueType) The types of the values.

Value

ValueTypes

Index