

Lazy evaluation in Haskell

exploring some mental models and implementations

Takenobu T.

Lazy,... zzz

..., It's fun.

NOTE

- Meaning of terms are different by communities.
- There are a lot of good documents. Please see also references.
- This is written for GHC's Haskell.

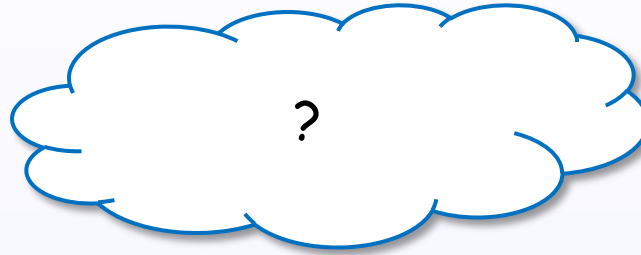
Contents

- Introduction
- Evaluations
- Expression in Haskell
- Constructor, WHNF, Thunk
- Evaluation in Haskell
- Control the evaluation in Haskell
- References

Introduction

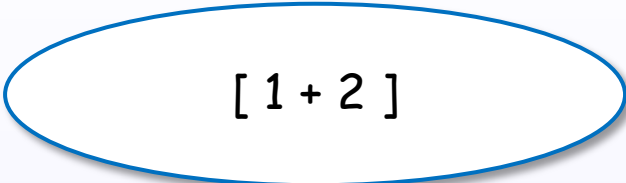
What is an expression?

An expression



An expression denotes a value

An expression



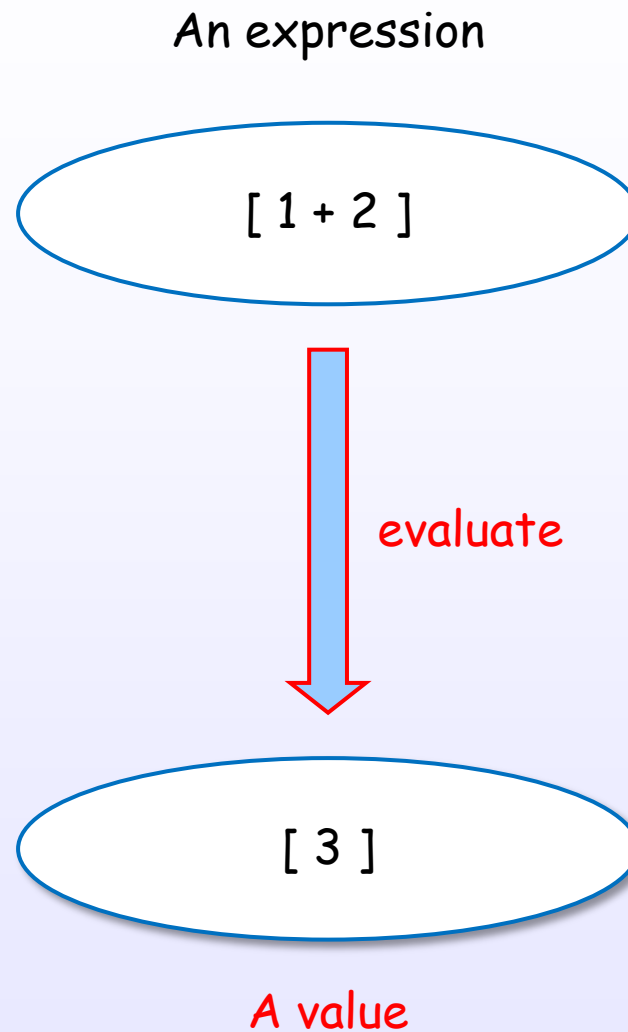
[1 + 2]

[HR2010]

[Bird, Chapter 2]

References : [1]

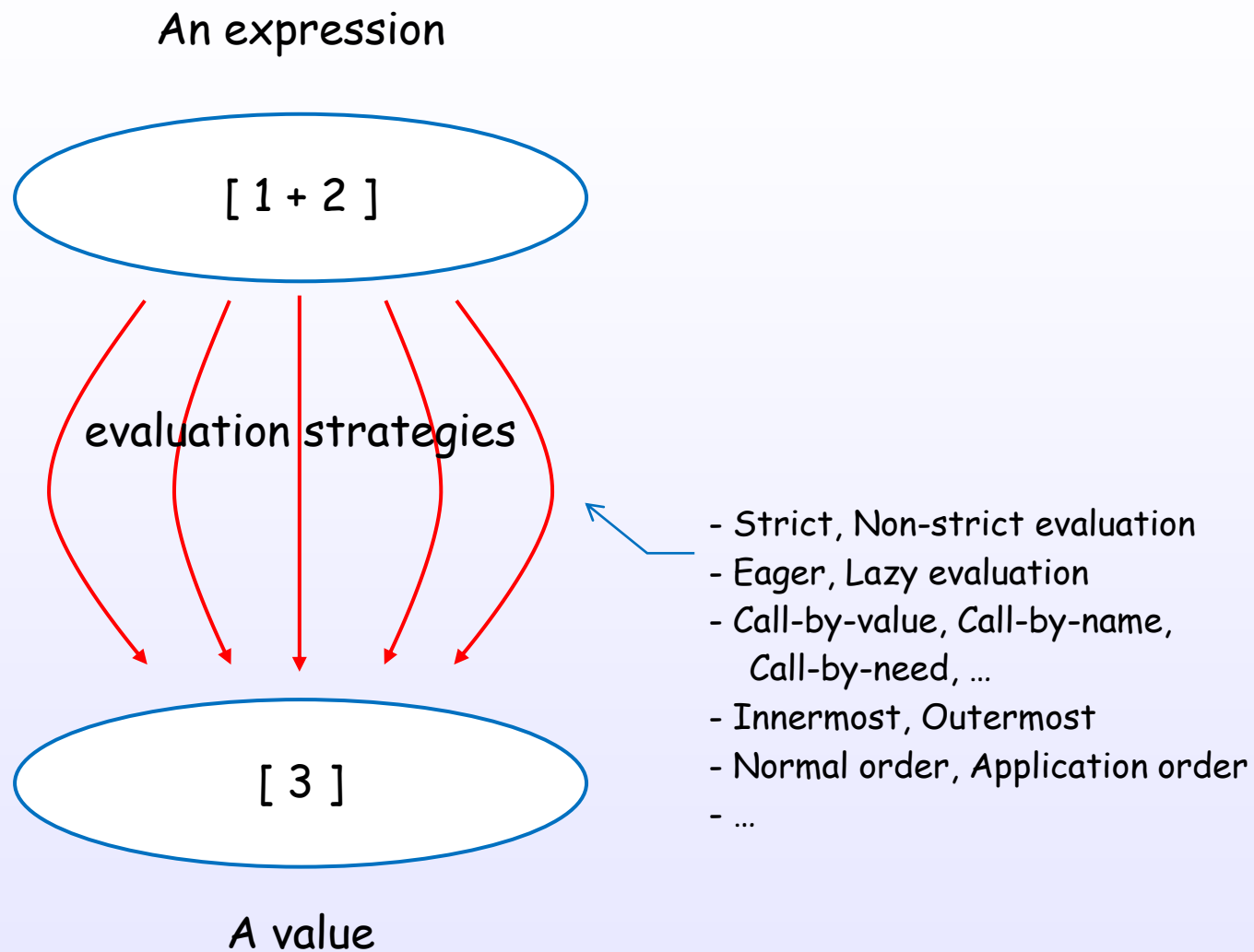
An expression evaluates to a value



[HR2010]

[Bird, Chapter 2]

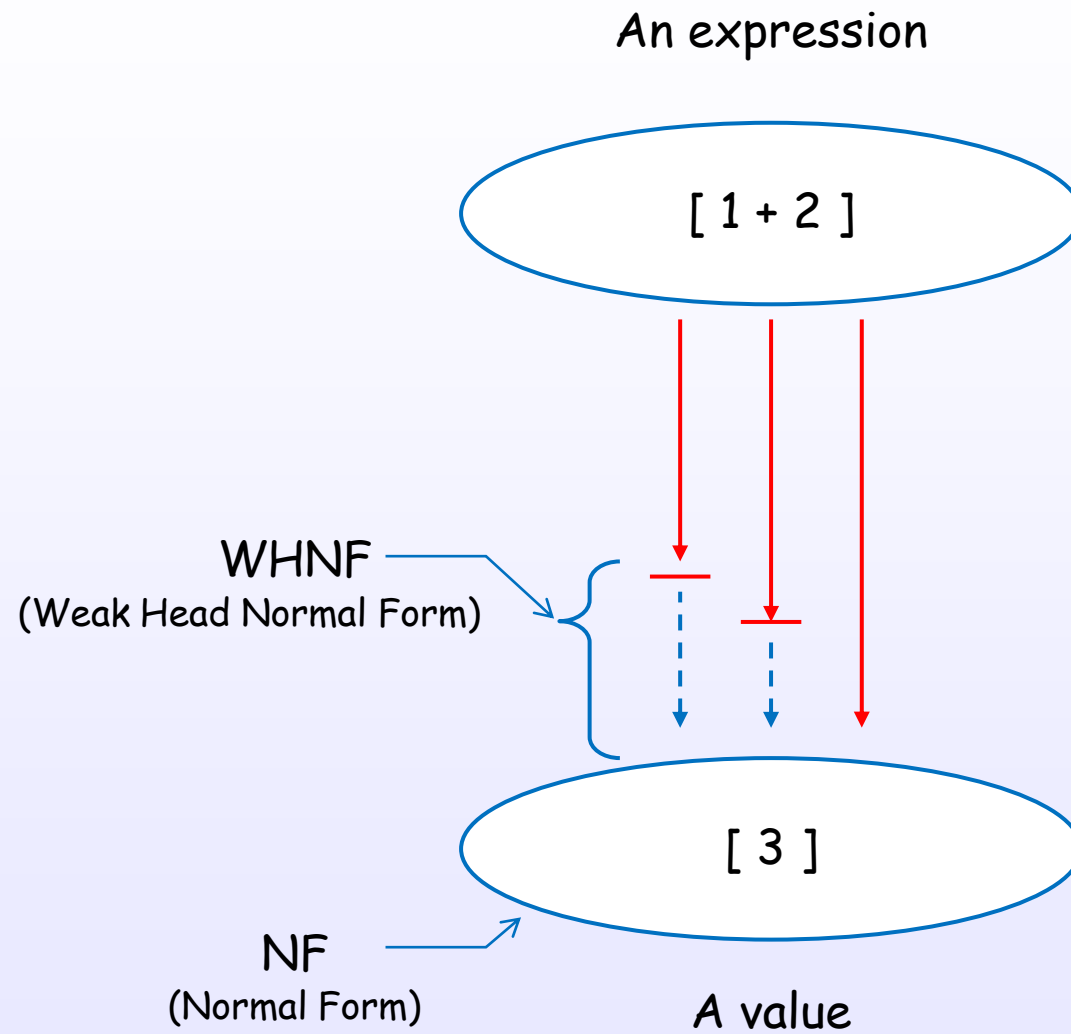
There are many evaluation approaches



[Bird, Chapter 2, 7]

[TAPL, Chapter 3]

There are many evaluation levels



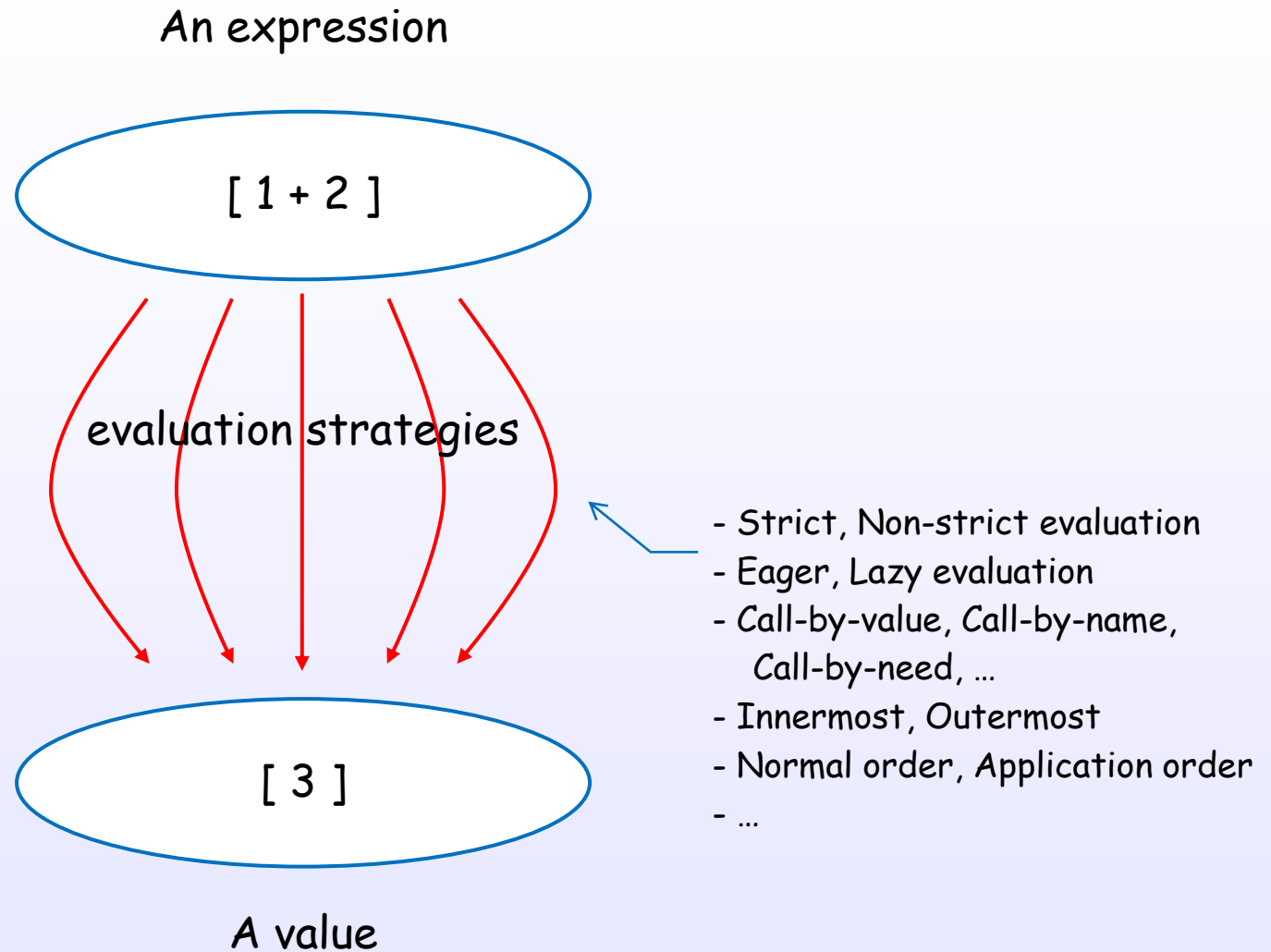
[Terei]

[Bird, Chapter 2, 7]

[TAPL, Chapter 3]

Evaluations

There are many evaluation approaches



[Bird, Chapter 2, 7]

[TAPL, Chapter 3]

evaluation layers

denotational semantics

evaluation strategy

evaluation implementation

evaluation layers

denotational
semantics

Strict semantics

Non-Strict semantics

evaluation
strategy

Eager evaluation
(Strict evaluation)

Nondeterministic
evaluation

Lazy evaluation
(Non-strict evaluation)

...

Call-by-Value

Call-by-Name

Call-by-Need

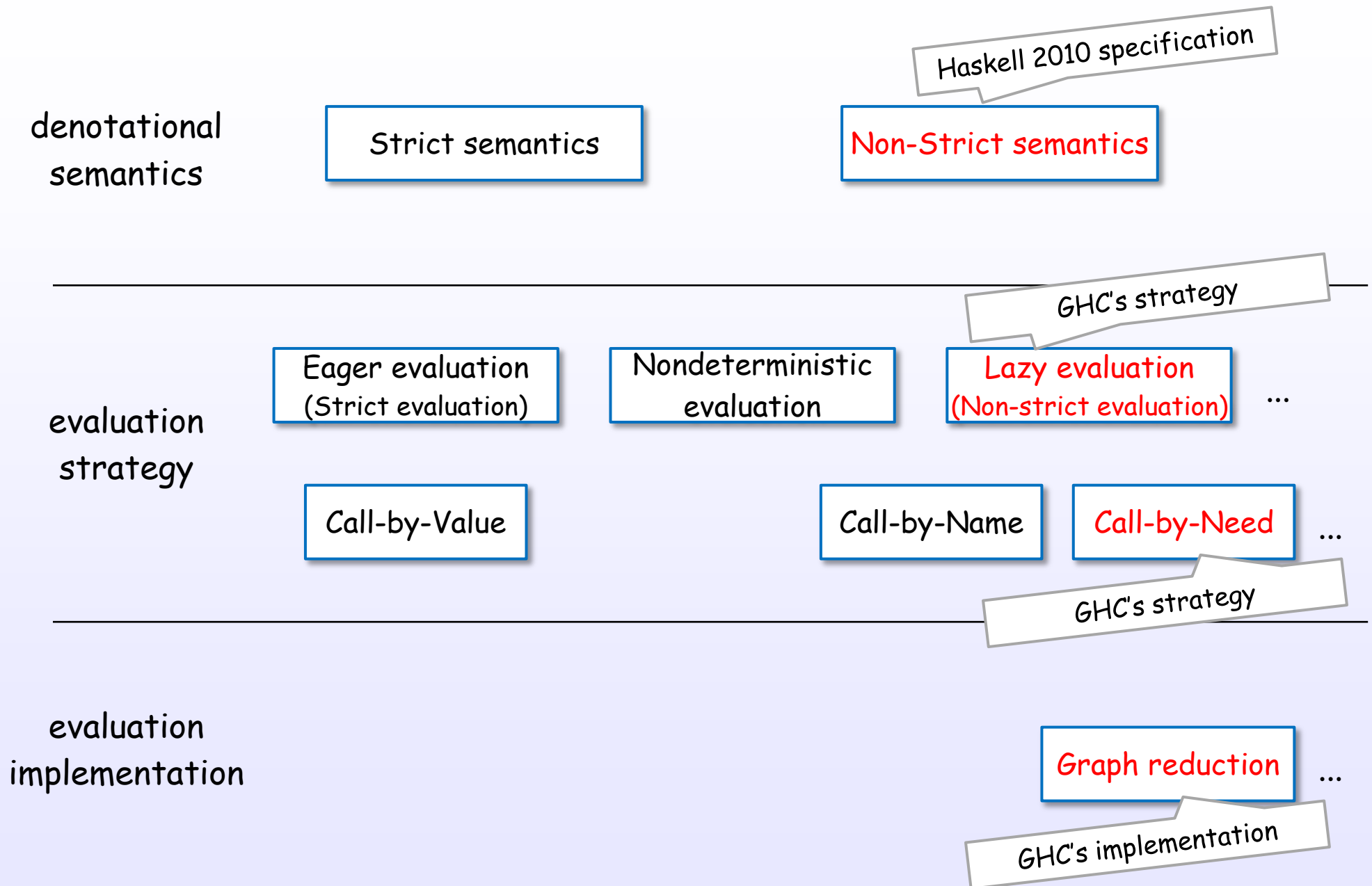
...

evaluation
implementation

Graph reduction

...

evaluation layers for GHC's Haskell



simple example of both evaluations

Eager evaluation (Strict evaluation)

C, JavaScript, Python,
OCaml, Scheme, ...

square (1 + 2)



argument
evaluation
first

square (3)



3 * 3



9

Lazy evaluation (Non-strict evaluation)

Haskell (GHC), ...

square (1 + 2)



apply
first

(1 + 2) * (1 + 2)



(3) * (3)



9

[Bird]
[Hutton]

simple example of both evaluations

Eager evaluation
(Strict evaluation)

square (1 + 2)



square (3)



3 * 3



9

argument
evaluated

Lazy evaluation
(Non-strict evaluation)

square (1 + 2)



(1 + 2) * (1 + 2)



(3) * (3)



9

argument
evaluation
delayed !

[Bird]
[Hutton]

Expression

An expression denotes a value

Just 5

$1 + 2$

$[1, 2, 3]$

take 5 xs

$\lambda x \rightarrow x + 1$

7

$\nexists x \rightarrow x + 1$

$\lambda x \perp \rightarrow x + 1$

[HR2010]

[Bird, Chapter 2]

What are expressions in Haskell

∀x → x + 1

let

if

case

do

Just 5

f a

7

What are expressions in Haskell

Haskell 2010 Language Report

<i>exp</i>	→ <i>infixexp</i> : : [<i>context</i> =>] <i>type</i> <i>infixexp</i>	(expression type signature)
<i>infixexp</i>	→ <i>lexp</i> <i>qop</i> <i>infixexp</i> - <i>infixexp</i> <i>lexp</i>	(infix operator application) (prefix negation)
<i>lexp</i>	→ \ <i>apat</i> ₁ ... <i>apat</i> _{<i>n</i>} -> <i>exp</i> let <i>decls</i> in <i>exp</i> if <i>exp</i> [;] then <i>exp</i> [;] else <i>exp</i> case <i>exp</i> of { <i>alts</i> } do { <i>stmts</i> } <i>fexp</i>	(lambda abstraction, $n \geq 1$) (let expression) (conditional) (case expression) (do expression)
<i>fexp</i>	→ [<i>fexp</i>] <i>aexp</i>	(function application)
<i>aexp</i>	→ <i>qvar</i> <i>gcon</i> <i>literal</i> (<i>exp</i>) (<i>exp</i> ₁ , ... , <i>exp</i> _{<i>k</i>}) [<i>exp</i> ₁ , ... , <i>exp</i> _{<i>k</i>}] [<i>exp</i> ₁ [, <i>exp</i> ₂] .. [<i>exp</i> ₃]] [<i>exp</i> <i>qual</i> ₁ , ... , <i>qual</i> _{<i>n</i>}] (<i>infixexp</i> <i>qop</i>) (<i>qop</i> ₍₋₎ <i>infixexp</i>)	(variable) (general constructor) (parenthesized expression) (tuple, $k \geq 2$) (list, $k \geq 1$) (arithmetic sequence) (list comprehension, $n \geq 1$) (left section) (right section)
	 <i>qcon</i> { <i>fbind</i> ₁ , ... , <i>fbind</i> _{<i>n</i>} }	(labeled construction, $n \geq 0$)
	 <i>aexp</i> _(<i>qcon</i>) { <i>fbind</i> ₁ , ... , <i>fbind</i> _{<i>n</i>} }	(labeled update, $n \geq 1$)

[HR2010]

Expressions examples

Constructor

priority

Evaluation

What is a value?

When? What extent?

Evaluation strategy

[Bird, Chapter 7]

[Hutton, Chapter 8]

[TAPL, Chapter 3]

References : [1]

[4]

normal form:

an expression without an redexes

head normal form:

an expression where the top level (head) is neither a redex NOR
a lambda abstraction with a reducible body

weak head normal form:

an expression where the top level (head) isn't a redex

[Terei]

[4]

evaluation strategies:

call-by-value: arguments evaluated before function entered (copied)

call-by-name: arguments passed unevaluated

call-by-need: arguments passed unevaluated but an expression is only evaluated once (sharing)

no-strict evaluation Vs. lazy evaluation:

non-strict: Includes both call-by-name and call-by-need, general term for evaluation strategies that don't evaluate arguments before entering a function

lazy evaluation: Specific type of non-strict evaluation. Uses call-by-need (for sharing).

[Terei]

Pattern match

[CIS194]

Tree, Graph

a expression

AST

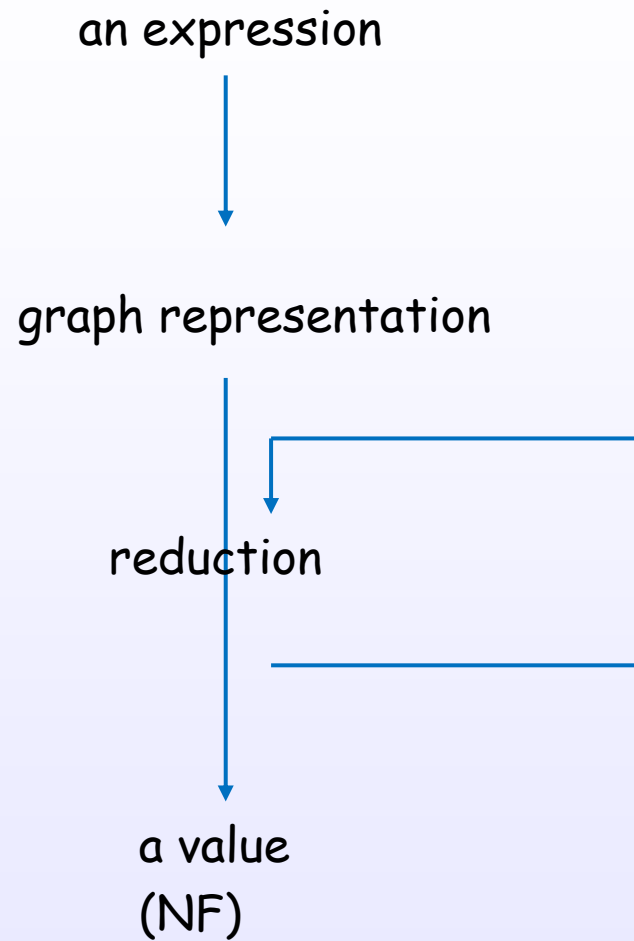
Tree

Graph

Shared Term

Lazy

evaluation, reduction



Thunk

Bottom

domain

co-domain

defined

undefined

$$f \perp = \perp$$

[Bird, Chapter 2]

Strictness, Bottom

[Bird, Chapter 2]

References : [1]

Layer

Non-strictness

$$f \perp = \perp$$

Lazy evaluation

Graph reduction

STG machine

Layer

Haskell semantics

take 5 [1..10]

internal representation

graph

STG semantics

heap object

STG machine

Evaluation in Haskell (GHC)

Evaluation in Haskell (GHC)

STG heap objects

language

Just 5

implementation

heap object

How to control the evaluation

control

case pattern match

seq

deepseq

!

IO

References

References

- [1] Haskell 2010 Language Report
<https://www.haskell.org/definition/haskell2010.pdf>
- [2] The Glorious Glasgow Haskell Compilation System (GHC user's guide)
https://downloads.haskell.org/~ghc/latest/docs/users_guide.pdf
- [3] Thinking Functionally with Haskell (IFPH 3rd edition)
<http://www.cs.ox.ac.uk/publications/books/functional/>
- [4] Types and Programming Languages
<https://mitpress.mit.edu/books/types-and-programming-languages>
- [5] A Haskell Compiler
<http://www.scs.stanford.edu/11au-cs240h/notes/ghc-slides.html>
[http://www.scs.stanford.edu/11au-cs240h/notes/ghc-slides.html#\(11\)](http://www.scs.stanford.edu/11au-cs240h/notes/ghc-slides.html#(11))
- [6] Being Lazy with Class
<http://www.seas.upenn.edu/~cis194/lectures/06-laziness.html>
- [7] The Incomplete Guide to Lazy Evaluation (in Haskell)
<https://hackhands.com/guide-lazy-evaluation-haskell/>
- [8] Programming in Haskell
<https://www.cs.nott.ac.uk/~gmh/book.html>
- [9] Parallel and Concurrent Programming in Haskell
<http://chimera.labs.oreilly.com/books/1230000000929/ch02.html>
- [10] Real World Haskell
<http://book.realworldhaskell.org/read/profiling-and-optimization.html>

References

- [11] Laziness
<http://dev.stephendiehl.com/hask/#laziness>
- [12] Evaluation on the Haskell Heap
<http://blog.ezyang.com/2011/04/evaluation-on-the-haskell-heap/>
- [13] How to force a list
<https://ro-che.info/articles/2015-05-28-force-list>
- [14] Haskell/Lazy evaluation
https://wiki.haskell.org/Haskell/Lazy_evaluation
- [15] Lazy evaluation
https://wiki.haskell.org/Lazy_evaluation
- [16] Lazy vs. non-strict
https://wiki.haskell.org/Lazy_vs._non-strict
- [17] Haskell/Denotational semantics
https://en.wikibooks.org/wiki/Haskell/Denotational_semantics
- [18] Haskell/Graph reduction
https://en.wikibooks.org/wiki/Haskell/Graph_reduction

References

- [19] Implementing lazy functional languages on stock hardware: the Spineless Tagless G -machine Version 2.5
<http://research.microsoft.com/en-us/um/people/simonpj/Papers/spineless-tagless-gmachine.ps.gz>
- [20] Making a Fast Curry Push/Enter vs Eval/Apply for Higher-order Languages
<http://research.microsoft.com/en-us/um/people/simonpj/papers/eval-apply/>
- [21] I know kung fu: learning STG by example
<https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/GeneratedCode>
- [22] GHC Commentary: The Layout of Heap Objects
<https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/HeapObjects>
- [23] GHC illustrated
http://takenobu-hs.github.io/downloads/haskell_ghc_illustrated.pdf

