Lazy evaluation in Haskell

exploring some mental models and implementations

Takenobu T.

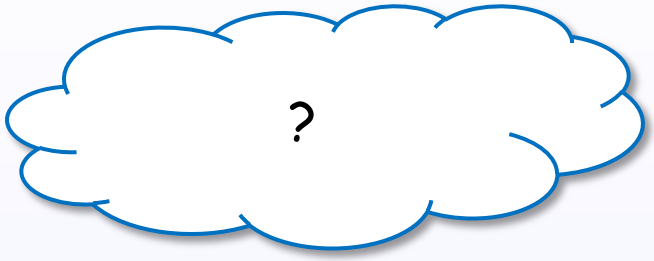
Rev. 0.00.0
WIP

Contents

Introduction

What is an expression?

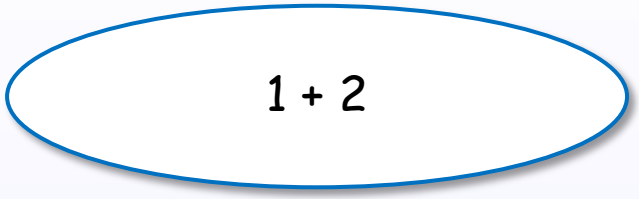
An expression



References : [1]

An expression denotes a value

An expression

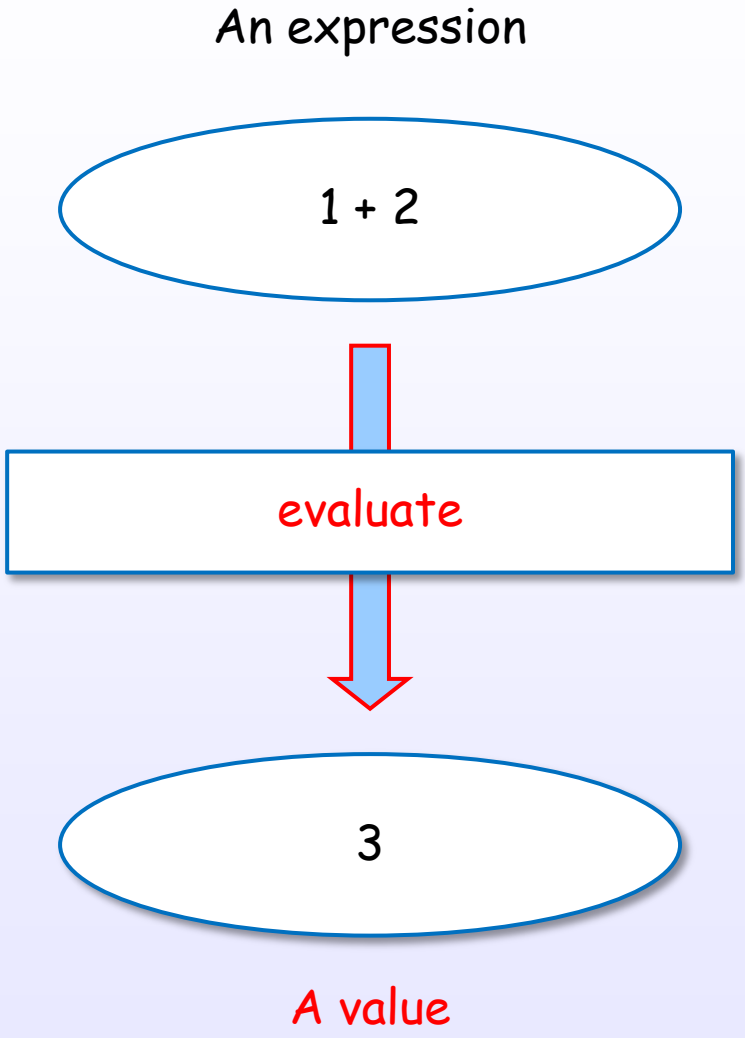


[HR2010]

[Bird, Chapter 2]

References : [1]

An expression evaluates to a value

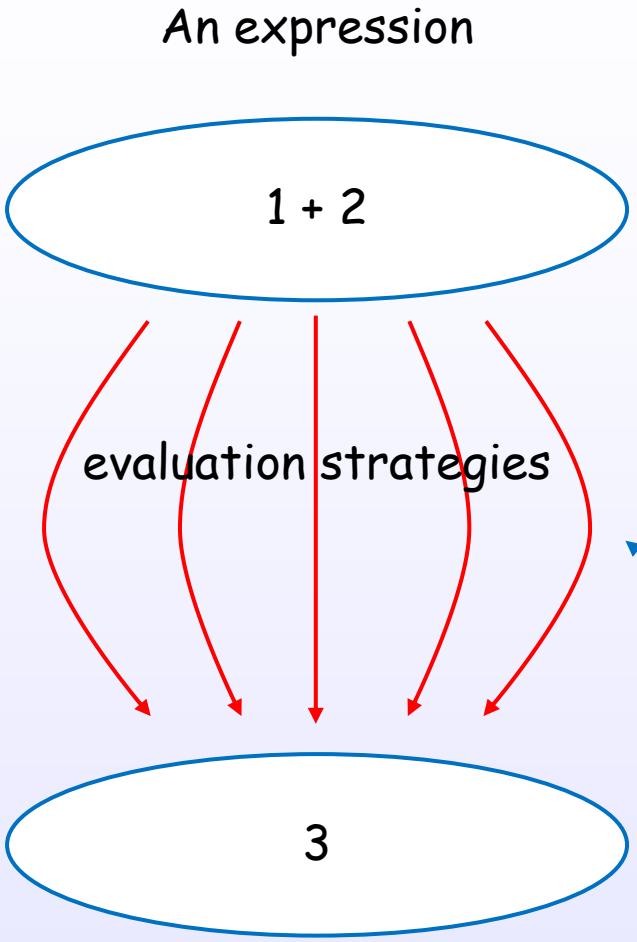


[HR2010]

[Bird, Chapter 2]

References : [1]

There are many evaluation approaches



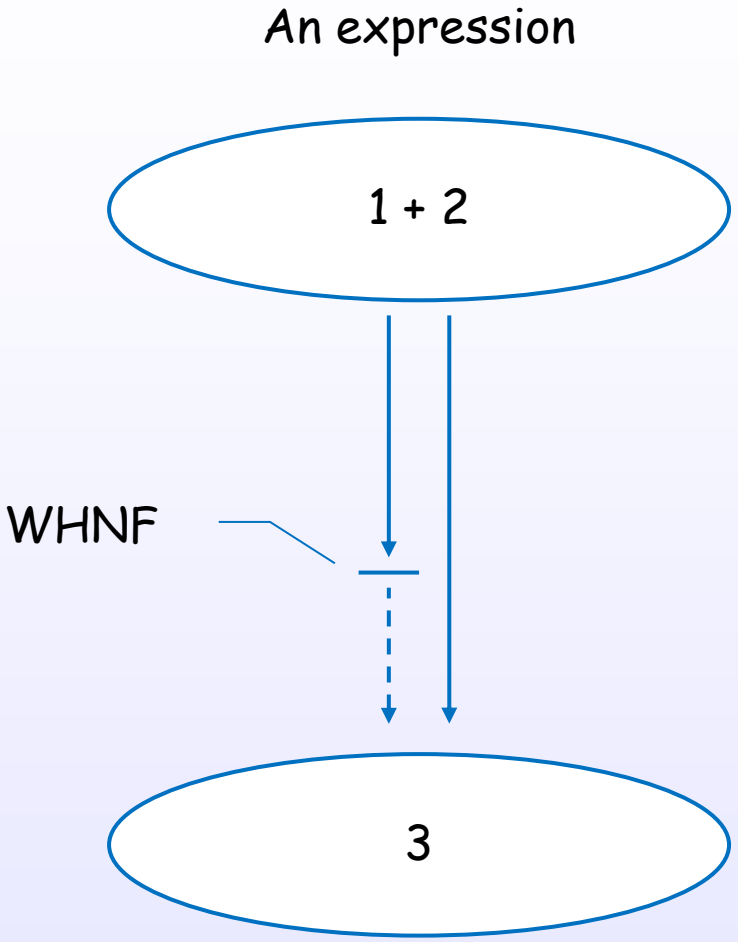
- Strict, Non-strict evaluation
- Eager, Lazy evaluation
- Call-by-value, Call-by-name, Call-by-need, …
- Innermost, Outermost
- Normal order, Application order
- …

[Bird, Chapter 2, 7]

[TAPL, Chapter 3]

References : [1]

What extent



[Terei]

[Bird, Chapter 2, 7]

[TAPL, Chapter 3]

References : [1]

Expression

An expression denotes a value

1 + 2

Just 5

[1, 2, 3]

take 5 xs

λx -> x + 1

7

¥x -> x + 1

λx ⊥ -> x + 1

[HR2010]

[Bird, Chapter 2]

References : [1]

What are expressions in Haskell

¥x -> x + 1

let

if

case

do

Just 5

7

f a

References : [1]

What are expressions in Haskell

Haskell 2010 Language Report

exp	→	*infixexp* `::` [*context* `=>`] *type*	(expression type signature)
	|	*infixexp*	
infixexp	→	*lexp qop infixexp*	(infix operator application)
	|	`-` *infixexp*	(prefix negation)
	|	*lexp*	
lexp	→	`\` $apat_1$ ... $apat_n$ `->` *exp*	(lambda abstraction, $n \geq 1$)
	|	`let` *decls* `in` *exp*	(let expression)
	|	`if` *exp* [`;`] `then` *exp* [`;`] `else` *exp*	(conditional)
	|	`case` *exp* `of` `{` *alts* `}`	(case expression)
	|	`do` `{` *stmts* `}`	(do expression)
	|	*fexp*	
fexp	→	[*fexp*] *aexp*	(function application)
aexp	→	*qvar*	(variable)
	|	*gcon*	(general constructor)
	|	*literal*	
	|	`(` *exp* `)`	(parenthesized expression)
	|	`(` exp_1 `,` ... `,` exp_k `)`	(tuple, $k \geq 2$)
	|	`[` exp_1 `,` ... `,` exp_k `]`	(list, $k \geq 1$)
	|	`[` exp_1 [`,` exp_2] `..` [exp_3] `]`	(arithmetic sequence)
	|	`[` *exp* `|` $qual_1$ `,` ... `,` $qual_n$ `]`	(list comprehension, $n \geq 1$)
	|	`(` *infixexp qop* `)`	(left section)
	|	`(` $qop_{\langle - \rangle}$ *infixexp* `)`	(right section)
	|	*qcon* `{` $fbind_1$ `,` ... `,` $fbind_n$ `}`	(labeled construction, $n \geq 0$)
	|	$aexp_{\langle qcon \rangle}$ `{` $fbind_1$ `,` ... `,` $fbind_n$ `}`	(labeled update, $n \geq 1$)

[HR2010]

References : [1]

Expressions examples

References : [1]

Constructor

References : [1]

priority

References : [1]

Evaluation

What is a value?

References : [1]

When? What extent?

References : [1]

Evaluation strategy

[Bird, Chapter 7]

[Hutton, Chapter 8]

[TAPL, Chapter 3]

References : [1]

WHNF

[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]

References : [1]

WHNF

[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]

References : [1]

Pattern match

[CIS194]

References : [1]

Tree, Graph

a expression

AST

Tree

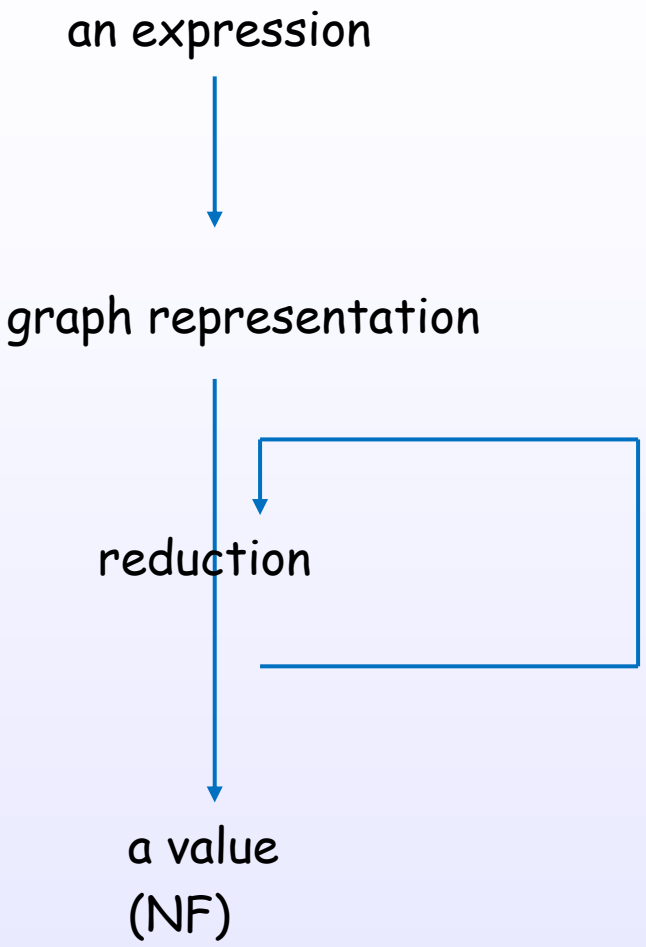
Graph

Shared Term

Lazy

References : [1]

evaluation, reduction



References : [1]

Thunk

References : [1]

Bottom

domain

co-domain

defined

undefined

$f \perp = \perp$

[Bird, Chapter 2]

References : [1]

Strictness, Bottom

[Bird, Chapter 2]

References : [1]

Layer

Non-strictness $f \perp = \perp$

Lazy evaluation

Graph reduction

STG machine

References : [1]

Layer

Haskell semantics	take 5 [1..10]
internal representation	graph
STG semantics	heap object
STG machine	

References : [1]

Evaluation in Haskell (GHC)

Evaluation in Haskell (GHC)

References : [1]

STG heap objects

language	Just 5
implementation	heap object

References : [1]

How to control the evaluation

control

case pattern match

seq

deepseq

!

IO

References : [1]

References

References

[1] Haskell 2010 Language Report
https://www.haskell.org/definition/haskell2010.pdf

[2] Thinking Functionally with Haskell (IFPH 3rd edition)
http://www.cs.ox.ac.uk/publications/books/functional/

[3] Types and Programming Languages
https://mitpress.mit.edu/books/types-and-programming-languages

[4] 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#(12)

[5] Being Lazy with Class
http://www.seas.upenn.edu/~cis194/lectures/06-laziness.html

[6] Evaluation on the Haskell Heap
http://blog.ezyang.com/2011/04/evaluation-on-the-haskell-heap/

[7] Programming in Haskell
https://www.cs.nott.ac.uk/~gmh/book.html

[8] Parallel and Concurrent Programming in Haskell
http://chimera.labs.oreilly.com/books/1230000000929
http://chimera.labs.oreilly.com/books/1230000000929/ch02.html

[9] Real World Haskell
http://book.realworldhaskell.org/
http://book.realworldhaskell.org/read/profiling-and-optimization.html

References

[10] Lazy evaluation
https://wiki.haskell.org/Lazy_evaluation

[11] Haskell/Lazy evaluation
https://wiki.haskell.org/Haskell/Lazy_evaluation

[12] Lazy vs. non-strict
https://wiki.haskell.org/Lazy_vs._non-strict

[13] The Incomplete Guide to Lazy Evaluation (in Haskell)
https://hackhands.com/guide-lazy-evaluation-haskell/

[14] Laziness
http://dev.stephendiehl.com/hask/#laziness

[15] How to force a list
https://ro-che.info/articles/2015-05-28-force-list

[16] Haskell/Graph reduction
https://en.wikibooks.org/wiki/Haskell/Graph_reduction

References

[17] 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

[18] 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/

[19] I know kung fu: learning STG by example
https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/GeneratedCode

[20] GHC illustrated
http://takenobu-hs.github.io/downloads/haskell_ghc_illustrated.pdf