50001 - Algorithm Analysis and Design - Lecture 2

Oliver Killane

12/11/21

Lecture Recording

Lecture recording is available here

Evaluation & Cost Models

```
minimum :: [Int] -> Int
minimum = head . isort
```

When analysing the cost of **minimum** we must consider how the function is evaluated.

For example we could shortcut the once **isort** has determined the first element (the minimum) of the list.

Cost Model

A model to determine the time taken to execute a program.

The model assigns cost to different operations (e.g comparisons, calls, memory reads/writes)

A very generalised cost model assigns cost based on the number of **reductions** required to evaluate a program.

Small While Language

We can define a small language of expressions as follows:

$$e ::= x \mid k \mid f\ e_1 \dots e_n \mid \text{if } e \text{ then } e_1 \text{ else } e_2$$

where k means constant and x is the variable form.
Infix functions such as $+, -, \times$ are written normally, and are also expressions as they can be used in the form $(+)\ e_1\ e_2$.

There are also several primitive constants: $True, False, 0, 1, 2, \dots$
List constants and operations are also primitive: $[], (:), null, head, tail$

Evaluation Order

- **Applicative Order** Strict evaluation
  The leftmost, innermost reducible expression is evaluated first.

e.g for $fst(3 \times 2, 1 + 2)$
$fst(3 \times 2, 1 + 2)$
$\rightsquigarrow$ { Definition of $\times$ }
$fst(6, 1 + 2)$
$\rightsquigarrow$ { Definition of $+$ }
$fst(6, 3)$
$\rightsquigarrow$ { Definition of fst }
6

- **Normal Order** Lazy evaluation
The leftmost outer reducible is evaluated first. Efectively evaluating the function before its arguments.
e.g for $fst(0, 1 + 2)$
$fst(3 \times 2, 1 + 2)$
$\rightsquigarrow$ { Definition of fst }
3×2
$\rightsquigarrow$ { Definition of $\times$ }
6

For a given program, if **applicative** and **normal** terminate, then they produce the same value in normal form.

However there are some programs where **normal** evaluation terminates, but **applicative** will not.

Applicative	**Normal**
$fst(0, \text{crazy nonesense})$	$fst(0, \text{crazy nonesense})$
$\rightsquigarrow$ { By lack of definition for crazy nonesense }	$\rightsquigarrow$ { Definition of fst }
$CRASH!$	0

The program may by syntactically correct, but have an error such as zero-division which will not be evaluated and hence not result in improper termination under **normal** order.

$$\text{Applicative Terminates} \Rightarrow \text{Normal Terminates}$$

Cost Model for Small While

We can evaluate a cost model for the small while language by creating a function T to assign cost to expressions.

Type	**Function**	**Explanation**
non-primitive function	$f\ a_1\ \ldots\ a_n = e$ $T(f)\ a_1\ \ldots\ a_n = T(e) + 1$	Given we have already computed all argument, the cost of the function is the cost of the expression it produces, and a single call.
primitive function	$T(f)\ x \ldots\ x_n = 0$	Primitive functions are assumed to be free.
Variable	$T(x) = 0$	accessing variables is free.
Application	$T(f\ e_1\ \ldots\ e_n) = T(f)\ e_1\ \ldots\ e_n + T(e_1) + \cdots + T(e_n)$	When applying a function we must consider both its cost, and the cost of all argument expressions.
Conditional	$T(\text{if } p \text{ then } e_1 \text{ else } e_2) = T(p) + \text{if } p \text{ then } T(e_1) \text{ else } T(e_2)$	Cost of condition and of the resulting expression.

Cost Model Example

Given the function:

$$mul\ m\ n = \text{if } m = 0 \text{ then } 0 \text{ else } n + mul\ (m-1)\ n$$

Evaluate $T(mul\ 3\ 100)$

(1)	$mul\ 3\ 100$	
(2)	$T(\text{if } 3 = 0 \text{ then } 0 \text{ else } 100 + mul\ (3-1)\ 100) + 1$	By Rule for non-primitive functions
(3)	$T(3 = 0) + T(100 + mul\ (3-1)\ 100) + 1$	By rule for conditionals
(4)	$0 + T(100 + mul\ (3-1)\ 100) + 1$	By primitive functions
(5)	$T(+)(100\ mul\ (3-1)\ 100) + T(100) + T(mul\ (3-1)\ 100) + 1$	By application rule
(6)	$0 + T(100) + T(mul\ (3-1)\ 100) + 1$	By rule for primitive functions
(7)	$0 + T(mul\ (3-1)\ 100) + 1$	By rule for constants
(8)	$T(mul)\ (3-1)\ 100 + T(3-1) + T(100) + 1$	By application rule
(9)	$T(mul)\ (3-1)\ 100 + T(-)3\ 1 + T(100) + 1$	By application rule
(10)	$T(mul)\ 2\ 100 + 1$	By application rule
(11)	$T(\text{if } 2 = 0 \text{ then } 0 \text{ else } 100 + mul\ (2-1)\ 100) + 1 + 1$	By Rule for non-primitive functions
(12)	$T(2 = 0) + T(100 + mul\ (2-1)\ 100) + 2$	By rule for conditionals
(13)	$0 + T(100 + mul\ (2-1)\ 100) + 2$	By primitive functions
(14)	$T(+)(100\ mul\ (2-1)\ 100) + T(100) + T(mul\ (2-1)\ 100) + 2$	By application rule
(15)	$0 + T(100) + T(mul\ (2-1)\ 100) + 2$	By rule for primitive functions
(16)	$0 + T(mul\ (2-1)\ 100) + 2$	By rule for constants
(17)	$T(mul)\ (2-1)\ 100 + T(2-1) + T(100) + 2$	By application rule
(18)	$T(mul)\ (2-1)\ 100 + T(-)2\ 1 + T(100) + 2$	By application rule
(19)	$T(mul)\ 1\ 100 + 2$	By application rule
(20)	$T(\text{if } 1 = 0 \text{ then } 0 \text{ else } 100 + mul\ (1-1)\ 100) + 2 + 1$	By Rule for non-primitive functions
(21)	$T(2 = 0) + T(100 + mul\ (1-1)\ 100) + 3$	By rule for conditionals
(22)	$0 + T(100 + mul\ (1-1)\ 100) + 3$	By primitive functions
(23)	$T(+)(100\ mul\ (1-1)\ 100) + T(100) + T(mul\ (1-1)\ 100) + 3$	By application rule
(24)	$0 + T(100) + T(mul\ (1-1)\ 100) + 3$	By rule for primitive functions
(25)	$0 + T(mul\ (1-1)\ 100) + 3$	By rule for constants
(26)	$T(mul)\ (1-1)\ 100 + T(1-1) + T(100) + 3$	By application rule
(27)	$T(mul)\ (1-1)\ 100 + T(-)2\ 1 + T(100) + 3$	By application rule
(28)	$T(mul)\ 0\ 100 + 3$	By application rule
(29)	$T(\text{if } 0 = 0 \text{ then } 0 \text{ else } 100 + mul\ (1-1)\ 100) + 3 + 1$	By Rule for non-primitive functions
(30)	$T(0 = 0) + T(0) + 4$	By rule for conditionals
(31)	$0 + T(0) + 4$	By rule for primitive functions
(32)	$0 + 4$	By rule for variables
(33)	4	