λ EXPRESSIONS & CURRIED FUNCTIONS

This page is intentionally blank

☺

LAMBDA EXPRESSIONS

- Here we are expanding on a topic first mentioned in the *functions* chapter of early discrete mathematics

- *Lambda* (λ) expressions provide a more succinct way of specifying functions

- Thus the infinite function {(7,12),(8,13), ...} may be specified by

$$\lambda\, p : \mathbb{N} \mid p > 6 \bullet p+5$$

  - p is a *dummy* variable
  - the more usual expression might be:

$$\{\, p : \mathbb{N} \mid p > 6 \bullet p \mapsto p+5 \,\}$$

LAMBDA EXPRESSIONS

- The general form of a lambda expression is:

  λ *signature* | *predicate* • *term* (or *expression*)

- Or, if no constraining predicate is necessary:

  λ *signature* • *term*

- Using a λ expression to define a function (i.e. a mapping from *domain* to *range*)

  - the *argument* (**domain** value) is moulded by the form of *signature* | *constraining predicate*

  - the **range** value mapped from the argument is given by the *term* or *expression*

- A *lambda expression*, therefore, describes maplet/ordered pairs of general form:

  (*signature* | *constraining predicate*) ↦ *term*

  or ((*signature* | *constraining predicate*), *term*)

LAMBDA EXPRESSIONS

- The general form of a lambda expression may also be interpreted as: $\lambda\ schema\_text \bullet term$

- This form may be read as:

  the function evaluating to *term*, with an argument structure described by *schema_text*

- Another lambda expression example is:

  - $\lambda\ a,b,c : \mathbb{N} \mid a+b = c \bullet a^2+b^2+c^2$

    equivalent to:

    $\{\ a,b,c : \mathbb{N} \mid a+b = c \bullet (a,b,c) \mapsto a^2+b^2+c^2\ \}$

    which expands to:

    $\{((0,0,0),0),((0,1,1),2),((1,0,1),2),((1,1,2),6), \ldots\}$

LAMBDA EXPRESSIONS

- Other lambda expression examples:
  - $square == \lambda\, x : \mathbb{N} \bullet x^2$
  - $succ == \lambda\, n : \mathbb{N} \bullet n + 1$
  - $pred == \lambda\, n : \mathbb{N}_1 \bullet n - 1$
  - $id == \lambda\, q \bullet q$ (NB generic)
  - $max == \lambda\, m,n : \mathbb{N} \bullet m \geq n \Rightarrow max(m,n) = m$
    $\wedge\ m < n \Rightarrow max(m,n) = n$

- The functions defined above, may, of course, be defined in less abstract fashion. For example:

  - *max* behaviour may be defined by

    $$\forall\, m,n : \mathbb{N} \bullet m \geq n \Rightarrow max(m,n) = m \wedge m < n \Rightarrow max(m,n) = n$$

  - and *square* may be defined axiomatically by

    $$f : \mathbb{N} \rightarrow \mathbb{N}$$
    $$\forall\, n : \mathbb{N} \bullet f\, n = n^2$$

LAMBDA EXPRESSIONS

- So, why bother with yet more notation?
- Because
  - by stating the *max* **property** using $\forall$, *max* is **not** being declared as an independent mathematical object; rather we are merely asserting a certain requirement on the value of *max* when applied to arbitrary arguments *m* and *n*, whereas
  - the λ expression introduces a *functional object* subject to any of the usual functional operators (eg $⨾$)
- More simply:
  - the $\forall$ version makes a statement *about* the function's behaviour, whereas
  - the lambda expression *defines* the function entirely

LAMBDA EXPRESSIONS

- The main advantage of lambda notation is that it treats functions as mathematical objects in their own right

- By treating functions as "first-class citizens" they acquire all the rights and privileges of normal mathematical objects and can be used as operands in expressions

- Such an approach makes it possible to express properties of functions without reference to the base sets on which the function operates:

    - e.g. $\mathit{succ} \mathbin{⨾} \mathit{pred} = \mathbf{id}$

      which is equivalent to:

      $$\forall n : \mathbb{N} \bullet (n + 1) - 1 = n$$

LAMBDA EXPRESSIONS

- As seen above, we are able to write expressions such as $f \circ g$ where f and g are functions being manipulated directly by the $\circ$ operator

- $f \circ g$ is a completely different animal to expressions such as $f\ (2) + g\ (7)$ where the + operator acts on *values* derived by *function application*

- The economy of notation which lambda expressions facilitate, also promotes considerable power for the manipulation of complex forms

- This is shown by the λ expression for a function *sum* which takes as arguments two other functions f and g and returns another function that maps an integer x to the sum of the two function applications $f\ x$ and $g\ x$:

$$sum == \lambda\, f, g : \mathbb{N} \rightarrow \mathbb{N} \bullet (\lambda\, x : \mathbb{N} \bullet f\ x + g\ x)$$

LAMBDA EXPRESSIONS

- As a further example, consider functions *square* and *negate* respectively (both with signature of type $\mathbb{Z} \rightarrow \mathbb{Z}$):

    - $square == \lambda\, x : \mathbb{Z} \bullet x^2$
    - $negate == \lambda\, x : \mathbb{Z} \bullet -x$

- Then we can define a whole series of function applications of the general form $sum\ (f, g)\ x$

    e.g. $sum\ (square, square)\ x$
    $sum\ (square, negate)\ x$
    $sum\ (negate, square)\ x$
    $sum\ (negate, negate)\ x$

LAMBDA EXPRESSIONS

- Since a lambda expression denotes a function (albeit one that is, maybe, nameless), such an expression can, as mentioned above, be subject to *function application*

- A special case of one of the forms given above might be the application:

$$sum\ (\ square, square)\ 5$$

- We have:

$sum\ (square, square)$

$= = \lambda\, f, g : \mathbb{Z} \rightarrow \mathbb{Z} \bullet$

$(\lambda\ x{:}\mathbb{Z} \bullet f\, x + g\ x\ )\ (square, square)$

$= \lambda\ x{:}\mathbb{Z} \bullet square\ x + square\ x$

and, $sum\ (square, square)\ 5$

$= (\lambda\ x{:}\mathbb{Z} \bullet square\ x + square\ x)\ 5$

$= 25 + 25 = 50$

LAMBDA EXPRESSIONS

- For another example of function application, consider a function *count_occs* which returns the number of times a value occurs in a sequence:

$$count\_occs\ [X] == \lambda\ v : X;\ s : \mathbf{seq}\ X \bullet \#(\ s \rhd \{\ v\ \}\ )$$

- In the special case when the sequence is $\langle 1,2,3,2,1 \rangle$ and the value of interest is 1, function application yields:

$$(\lambda\ v{:}X;\ s{:}\mathbf{seq}\ X \bullet \#(s \rhd \{\ v\ \}))\ (1,\langle 1,2,3,2,1 \rangle)$$
$$= 2$$

because v is substituted by 1 and s is substituted by $\langle 1,2,3,2,1 \rangle$ in the *term* of the lambda expression

CURRIED FUNCTIONS

- Now suppose that f is a function where:

$$f : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$$

meaning that the input argument is now an *ordered pair* of integers and the function will yield another integer

- An example of such a function could be:

$$f == \lambda\ a,b : \mathbb{N} \bullet a+b+2ab$$

equivalent to:

$$f == \{\ a,b : \mathbb{N} \bullet (a,b) \mapsto a+b+2ab\}$$

- To evaluate what f generates for a particular ordered pair of values (say, a=5 and b=3) we could use function application:

$$f(5, 3) = 5+3+(2\times5\times3) = 5+3+30 = 38$$

CURRIED FUNCTIONS

- The above result could, also, be derived in stages:
  - first partially compute f substituting **only** the value for a: $f\ 5\ b = 5+b+(2\times5\times b) = 5+11b$;
  - and the outcome is itself a function g, say, of a single variable (b);
  - next compute $g\ 3$ to recover the same final result: $g\ 3 = 5+(11\times3) = 5+33 = 38$

- This incremental approach to evaluation is called *currying* (after logician *Haskell B Curry*)

- In the above example,
  - the original function was of type
    $$\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$$
  - the *curried* version had type
    $$\mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow \mathbb{Z})$$
    and this is written, more usually, as
    $$\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$$

CURRIED FUNCTIONS

- In general, we can always turn a function with n arguments into a function having (n-1) arguments by using function application for one of the arguments

- Lambda expressions are a convenient way of representing curried functions:

    - suppose *sum* is an infinite function which forms the sum of two natural numbers

    - $$\begin{aligned} sum == & \lambda\, m,n : \mathbb{N} \bullet m + n \\ = & \lambda\, m : \mathbb{N} \bullet \lambda\, n : \mathbb{N} \bullet m + n \\ = & \lambda\, m : \mathbb{N} \bullet (\lambda\, n : \mathbb{N} \bullet (m + n)) \end{aligned}$$

    - $$\begin{aligned} sum\ 6\ 9 = & (\lambda\, m : \mathbb{N} \bullet \lambda\, n : \mathbb{N} \bullet m + n)\ 6\ 9 \\ = & (\lambda\, m : \mathbb{N} \bullet (\lambda\, n : \mathbb{N} \bullet (m + n)\ 6)\ 9 \\ = & (\lambda\, m : \mathbb{N} \bullet m + 6)\ 9 \\ = & 9 + 6 = 15 \end{aligned}$$

CURRIED FUNCTION EXAMPLES

- An example which illustrates both lambda expressions and curried functions is the problem of specifying the *greatest common divisor* function, *gcd*

    - *The greatest common divisor (or, highest common factor) of two positive integers is the largest integer which exactly divides both of the given integers*

- We are concerned with the set $\mathbb{N}_1$ (i.e. $\mathbb{N} \setminus 0$)

- To facilitate the construction of a concise specification we first define the function *divisors* which maps any positive integer to the set of all its divisors:

$$divisors == \{ \quad (1,\{1\}),\ (2,\{1,2\}),\ (3,\{1,3\}),\ (4,\{1,2,4\}),\ (5,\{1,5\}),\ (6,\{1,2,3,6\}),\ (7,\{1,7\}),\ (8,\{1,2,4,8\}),\ \ldots \quad \}$$

CURRIED FUNCTION EXAMPLES

- Note that *divisors* cannot be empty (since 1 will always divide any number) and, more formally:

  $$divisors == \lambda\, k : \mathbb{N}_1 \bullet \{m, n : \mathbb{N}_1 \mid m \times n = k \bullet m\}$$

- Avoiding λ-notation we could, instead, create an axiomatic definition :

  $$\begin{array}{|l}
divisors_1 : \mathbb{N}_1 \rightarrow \mathbb{P}_1\, \mathbb{N}_1 \\
\hline
\forall\, k : \mathbb{N}_1 \bullet divisors_1 = \{m, n : \mathbb{N}_1 \mid m \times n = k \bullet m\}
\end{array}$$

- Also note that the "term" in the lambda expression could equally well be *n* (the other factor of the product):

  $$divisors_2 == \lambda\, k : \mathbb{N}_1 \bullet \{m, n : \mathbb{N}_1 \mid m \times n = k \bullet n\}$$

- *divisors*, $divisors_1$ and $divisors_2$ are equivalent

CURRIED FUNCTION EXAMPLES

- From the definition, the greatest common divisor (*gcd*), of a pair of positive integers is the largest divisor they have in common; so:

  $gcd == \lambda\, i, j : \mathbb{N}_1 \bullet max(divisors\ i \cap divisors\ j)$

  - but *max* should only be applied to a non-empty set; so
  - can we be certain $divisors\ i \cap divisors\ j \neq \{\}$?

- If gcd_1 is the curried version of *gcd*, we have:

  $gcd_1 == \lambda\, i : \mathbb{N}_1 \bullet (\lambda\, j : \mathbb{N}_1 \bullet max(divisors\ i \cap divisors\ j))$

CURRIED FUNCTION EXAMPLES

- Consider another example:
    - *head* is a function that takes a sequence of lines of text and a natural number n and returns the first n of the given lines; all lines in the given text are returned in the case when n exceeds the number of given lines

- The specification is not that precise and we make the following assumption:
    - Each *line* is a sequence of *characters* terminated by the *newline* character(s); a *newline* may not occur anywhere else in the sequence

CURRIED FUNCTION EXAMPLES

- A specification of *head* might then be:

  [CHAR] the set of all possible characters

  $line == \{ l : \mathbf{seq}(char \setminus \{ nl \}) \bullet l \frown \langle nl \rangle \}$

  with:

  $$\begin{array}{|l} head : \mathbb{N} \times \mathbf{seq}\ line \rightarrow \mathbf{seq}\ line \\ \hline \forall\ txt : \mathbf{seq}\ line;\ n : \mathbb{N} \bullet head\ (n, txt) = 1 \mathrel{..} n \lhd \end{array}$$

- Or, using a lambda expression:

  $$head == \lambda\ n : \mathbb{N};\ txt : \mathbf{seq}\ line \bullet 1 \mathrel{..} n \lhd txt$$

- Or, using a *curried* version, $head_1$:

  $$head_1 == \lambda\ n : \mathbb{N} \bullet (\lambda\ txt : \mathbf{seq}\ line \bullet 1 \mathrel{..} n \lhd txt)$$

CURRIED FUNCTION EXAMPLES

- Suppose we revisit the *count_occs* function:

$$count\_occs\ [X] == \lambda\ v : X; s : \mathbf{seq}\ X \bullet \#(\ s \rhd \{\ v\ \}\ )$$

- A generic definition, which does not use lambda expressions, might have the form:

$$[X]$$
$$count\_occs : X \times \mathbf{seq}\ X \rightarrow \mathbb{N}$$
$$\forall\ v : X; s : \mathbf{seq}\ X \bullet count\_occs\ (v, s) = \#(\ s \rhd \{\ v\ \}\ )$$

- This latter (non-λ) form has a corresponding curried version *curr_count_occs*:

$$[X]$$
$$curr\_count\_occs : X \rightarrow \mathbf{seq}\ X \rightarrow \mathbb{N}$$
$$\forall\ v : X; s : \mathbf{seq}\ X \bullet curr\_count\_occs\ v\ s = \#(\ s \rhd \{\ v\ \}\ )$$

CURRIED FUNCTION EXAMPLES

- It is important to realize that, although the overall effects are the same, *count_occs* and *curr_count_occs* are *different* functions (because the mappings are different!)

- The fact that we have different functions can be highlighted by expressing them with lambda notation:

    - *count_occs*[X] = =
      $\lambda\, v : X;\, s : \mathbf{seq}\ X \bullet \#(s \rhd \{\, v \,\})$

    - *curr_count_occs*[X] = =
      $\lambda\, v{:}X \bullet (\lambda\, s{:}\mathbf{seq}\ X \bullet \#(s \rhd \{\, v \,\})$

CURRIED FUNCTION EXAMPLES

- Suppose we wish to find the number of times that 0 occurs in a list of integers, *sequint*, say, where $sequint : \mathbf{seq}\ \mathbb{Z}$

- Using the two different functions, just derived, we would write the required terms as:

  - $count\_occs\ (0, sequint)$
    $= (\lambda\ v : \mathbb{Z};\ s : \mathbf{seq}\ \mathbb{Z} \bullet \#(s \rhd \{\ v\ \})(0, sequint)$
    $= \#(sequint \rhd \{\ 0\ \})$

  - $curr\_count\_occs\ \ 0\ sequint$
    $= (curr\_count\_occs\ \ 0) sequint$
    $= (\lambda v{:}\mathbb{Z} \bullet (\lambda s{:}\mathbf{seq}\ \mathbb{Z} \bullet \#(s \rhd \{v\}))\ 0)\ sequint$
    $= (\lambda\ s{:}\mathbf{seq}\ \mathbb{Z}\ \ \bullet \#(s \rhd \{\ 0\ \}))\ sequint$
    $= \#(sequint \rhd \{\ 0\ \})$

CURRIED FUNCTION EXAMPLES

- It is worth noting that, in the curried version, *curr_count_occs* 0 represents a function that counts *zeros* in any integer list; in full:

$$\lambda\, s{:}\mathbf{seq}\,\mathbb{Z} \bullet \#(s \rhd \{\ 0\ \})$$

- We can, therefore, define a function *count_occs_of_*0 by:

$$count\_occs\_of\_0 == curr\_count\_occs\ 0$$

- And, by extension, we can define a general counting function for such integer lists:

$$count\_occs\_of\_k == curr\_count\_occs\ k$$

- The uncurried version could not generate other more specialized functions in such an easy way

SUMMARY OF SYMBOLS

$f\,x$	*the function f applied to x*
$\lambda D \mid P \bullet E$	*function definition where P is a predicate constraining the values declared in D and E is an expression giving the function in terms of the values declared in D*

This page is intentionally blank
☺

EXERCISES

1. From book by *Ince*:

   (a) Give the expanded form (i.e. a set of ordered pairs) which corresponds to the function defined by: $\lambda\, x : \mathbb{N} \mid x < 3 \bullet x^2$ *(page 148)*

   (b) If *files* is the set: { upd, text, ed1, ed2, ed3, tax1, tax2 }
   and *size* is a partial function over *files* $\times\ \mathbb{N}$ with the value:
   { (upd, 45), (text, 175), (ed1, 105), (ed2, 95) }
   write down the values of the following expressions: *(page 149)*

   (i) $\lambda\, x : 3 \,..\, 15 \mid x^2 \leq 9 \bullet x$
   (ii) $2 \,..\, 4 \lhd (\lambda\, x : \mathbb{N} \mid x < 10 \bullet \mathrm{x})$
   (iii) $\lambda\, x : \mathbb{N} \mid x < 10 \bullet x^2$
   (iv) $\lambda\, x : \mathbf{ran}\ size \mid x > 53 \bullet x^2 + 10$
   (v) $\lambda\, x : \mathbb{N} \mid x \in \mathbf{ran}\ size \bullet x$
   (vi) $\{2, 3\} \lhd (\lambda\, x : \mathbb{N} \bullet x^2)$
   (vii) $size \mathbin{⨾} (\lambda\, x : \mathbb{N} \mid x \neq 100 \bullet x + 10)$
   (viii) $(\lambda\, x : \mathbb{N} \mid x \neq 5 \land x < 10 \bullet x) \cup (\lambda\, x : \mathbb{N} \mid x^2 = 16 \bullet x^3)$

2. If $sum == \lambda\, f, g : \mathbb{N} \rightarrow \mathbb{N} \bullet (\, \lambda\, x : \mathbb{N} \bullet f\, x + g\, x\,)$ what are the outcomes of the following applications:

   (a) *sum* (*square*, *negate*) 5;
   (b) *sum* (*negate*, *square*) 5;
   (c) *sum* (*negate*, *negate*) 5

3. Construct **generic** definitions of the following functions (see generic definition of *count_occs* in the handout), first without using λ expressions and then using λ expressions:

   elements_of (*s*) : which yields the elements of sequence *s* as a set
   delete_all (*x, s*) : which yields the sequence obtained by deleting from sequence *s* all instances of *x*

4. Explain the differences, if any, that exist between the following functions:

   (a) $\mathrm{f}_1 : \mathrm{A} \rightarrow \mathrm{B} \rightarrow \mathrm{C} \rightarrow \mathrm{D}$
   (b) $\mathrm{f}_2 : (\mathrm{A} \rightarrow \mathrm{B}) \rightarrow \mathrm{C} \rightarrow \mathrm{D}$
   (c) $\mathrm{f}_3 : \mathrm{A} \rightarrow (\mathrm{B} \rightarrow \mathrm{C}) \rightarrow \mathrm{D}$
   (d) $\mathrm{f}_4 : \mathrm{A} \rightarrow \mathrm{B} \rightarrow (\mathrm{C} \rightarrow \mathrm{D})$

5. Assume the existence of a function *first_pos_in* (*e*, *t*) which yields the **position** of the first occurrence of *e* in sequence *t* (for no occurrence of *e* in the sequence *t* the function yields zero).

   (a) Give a generic, non-λ, definition for function *del_first_occ* (*x*, *s*) which yields the sequence obtained by deleting from sequence *s* the first occurrence of *x* (if any).
   (b) Give an equivalent form using a λ expression.
   (c) What is the curried version of your answer to part (b)?

6. From the curried function in 5 above, derive a suitable definition for a function *del_first_occ_nullset* that deletes the first instance of the **empty set** from a list of sets of integers.

7. A model underlying GUI design in a particular environment is based around the following abstractions:

context	:	a graphical object (e.g. button, window, menu, etc) comprising part of the interactive system: an associated boolean function returns *true* if the mouse cursor is within context's graphical area; *false* otherwise
interface	:	the set of *contexts* making up its visual appearance
event	:	a user-triggered action that can occur at run-time (e.g. by clicking a "button")
command	:	an object describing the execution of some operation of the generated system - addresses *semantics* underlying the *interface*
state	:	defines the possibilities that are open to users at a given stage of a session
transition	:	is a value attached to a *command* and effectively defines one of the allowable outcomes if a *command* instance is executed

In particular, *state* is a function with *signature*:

$$\text{CONTEXT} \times \text{EVENT} \twoheadrightarrow\!\!\!\!\!\!\mapsto \text{COMMAND}$$

   (a) What is a curried version of the *state* function?
   (b) If the part of a *state* which is described by EVENT ⇸⇸ COMMAND (for a particular *context*) is called the *behaviour*, *state* may be viewed as having what signature?
   (c) What other signature is possible (though probably impractical) for *state*?