McMaster University
Department of Computing and Software
Musa Al-hassy

COMP SCI 3EA3 — Software Specification and Correctness

Feburary 4, 2017

Exercise 5.0 — Lists: the most beloved monoids in software

1. Every type X has a *free monoid* `(List X, ++, [])` consisting of lists/sequences of elements from X with binary operation being catenation of lists whose identity is the empty list `[]`:

   $$\mathsf{List} : \mathsf{Type} \to \mathsf{Type}$$

   Every element of `List X` is either the empty list or an element of `x : X` followed by another list `xs : List X` and this is denoted `x :: xs`; give a formal grammar for such a structure.

2. The catenation operation is informally specfied by

   $$(++) : \mathsf{List}\ X \to \mathsf{List} X \to \mathsf{List} X$$

   $$[x_0, x_1, \ldots, x_{n-1}] ++ [y_0, y_1, \ldots, y_{m-1}] = [x_0, x_1, \ldots, x_{n-1}, y_0, y_1, \ldots, y_{m-1}]$$

   use the induction/recursion principle of the grammar you defined to give a formal recursive/inductive definition of catenation.

3. Prove by induction that the empty list `[]` is the identity of catenation and catenation is itself associative; thereby showing we have a monoid.

4. Axiomatize lists and some of their properties in `Frama-C`.

5. Implement lists in C and implement catenation as well, then prove it correct in `Frama-C`.

6. Discuss how lists differ from stacks and from trees? What conventions or connotations are associated with each?

7. Research why these structures are known as "free monoids" —we'll return to this at a later time in the course.

Exercise 5.1 — Quantification: the return of the monoid!

1. Show that every bounded lattice $(L, \sqsubseteq, \sqcap, \sqcup, \top, \bot)$ gives rise to two monoids: $(L, \sqcap, \top)$ and $(L, \sqcup, \bot)$.
   What are the associated monoids when we consider the integers? What about when we consider programs? ★

2. For a monoid $(M, \oplus, e)$ and a function $f : \mathbb{N} \to M$ we may define a "value" $(\oplus i \mid a \leq i \leq b \bullet f\ i)$ called "the qunatifcation over f from a to b" —where $a, b : \mathbb{N}$ have $a \leq b$— informally defined by

   $$(\oplus i \mid a \leq i \leq b \bullet f\ i) = f\ a \oplus f(a+1) \oplus f(a+2) \oplus f(a+3) \oplus \cdots \oplus f\ b$$

   Give an inductive, axiomatic, definition of this value in-terms of a and b.
   Do so on paper, then do so in `Frama-C` ★.

3. In `Frama-C`, code and prove correct a program to sum the elements of an array.
   What is the sum of the first n-natural numbers?

4. In `Frama-C`, code and prove correct a program to product the elements of an array.
   What is the product of the first n-natural numbers?

5. Specify an algorithm generalizing the previous two —indeed addition and multiplication each produce a monoid ;)

   Mottos: "Quantification corresponds to loops" ; "Induction corresponds to recursion"

6. **Read** chapter 11 of the course text on a more general notion of quantifiers. ★

   Unlike the text, we will use Z notation for quantifiers, comprehensions, and other similar extensions. For example, we will write

   $$(\oplus \mathit{declaration} \mid \mathit{predicate} \bullet \mathit{term})$$

   whereas the text writes $\langle \oplus \mathit{declaration} : \mathit{predicate} : \mathit{term} \rangle$. Moreover, the "declaration" part, for us, is a semi-colon separated sequence of identifiers and their types, $\mathit{var} : \mathit{type}$ —we use colon for typing, which may be omitted when typing is clear from context.

   Note that "variable binding" is everywhere: $\int_0^1 f(x)dx$ from first-year Calculus, `int f(int x){ ... }` from C programming, and `let x = E in F` from Haskell all bind x as a "local variable".

7. **Memorise** the quantifier rules of §11.4 of the course text; we will use them often!

   How does each rule correspond to an equality of loop-programs?

8. CalcCheck (Quiz problems *may* be drawn from these particular exercises.)

   Log onto Avenue and try proving the theorems and solving the puzzles and problems taken from last term's comp sci 2dm3 class with Professor Wolfram Kahl, whose tool CalcCheck we will use to obtain immediate feedback on our proofs.

   "Avenue → Contents → Exercises (CS2DM3 Prof Kahl) → Exercises 9.1-9.3"

Please send any feedback to tool provider at `kahl@cas.mcmaster.ca`!

Exercise 5.2 — Posets: bringing order to life

Recall that a poset $(L, \sqsubseteq)$ is a type L endowed with a binary relation $\sqsubseteq$ that is reflexive, transitive, and antisymmetric. The first two axioms can jointly be replaced by either of the following equivalent and beautiful formulations.

$$\text{"indirect inclusion from } a\text{bove"}\colon\ x \sqsubseteq y \quad \equiv \quad (\forall a \bullet x \sqsubseteq a \Leftarrow y \sqsubseteq a)$$

$$\text{"indirect inclusion from } b\text{elow"}\colon\ x \sqsubseteq y \quad \equiv \quad (\forall b \bullet b \sqsubseteq x \Rightarrow b \sqsubseteq y)$$

The second reads: "x is contained in y precisely when items contained by x are also contained by y".

Examples include natural numbers with division as order, subsets of a set with set-inclusion as order, and types in an OOP language with sub-type relation as order.

1. Use the indirect principles to prove that the order is reflexive and transitive. ★
2. A *monotone map* is an "order preserving function". Define this formally.
3. Similar to the "retract setoid" of the previous sheet, formalise the notion of "pointwise order": an order on $(P, \sqsubseteq)$ induces an order $\dot{\sqsubseteq}$ on the function-space $X \to P$, for any X.
4. Formalise the notions of "maximal"/"top" '$\top$' and "minimal"/"bot" '$\bot$' elements in a poset.

   An common example is in mathematics when one wants to treat the infinity —the concept of growing without bound— as a number, one uses the 'extended real line'.

   A *bounded* poset is a poset with a top and bot; a *bounded* lattice is a lattice with a top and bot.

   (a) The importance of an order: do the 'extended integers' from Sheet 3 have a bottom element, or top element, or both?

   (b) Given any type X explain how we may endow it with fictitious bottom and top elements using the structure `Maybe (Maybe X)`?

   (c) A "zero element" in a monoid $(M, \oplus, e)$ is an element z with the property $x \otimes z = z = z \otimes x$ for any x. Prove that in a lattice, top is a zero element for join. What is the dual of the previous statement; prove it.

Exercise 5.3 — Galois Connections: approximate inverses

Recall that a *Galois Connection* "$L \dashv U$" between two posets $(C, \sqsubseteq_C)$ and $(S, \sqsubseteq_S)$ is a pair of functions $L : C \to S$ —from a "Complicated" domain to a "Simpler" domain— and $U : S \to C$ such that

$$\forall x, y \bullet L\ x \sqsubseteq_S y \quad \equiv \quad x \sqsubseteq_C U\ y$$

The functions L and U are known as the "adjoints" of the connection.

1. A *brute-force* algorithm to test whether two functions are Galois connected would be a double for-loop —looping over all x candidates, and for each such we then loop for all y candidates— and hence quadratic time. This can be inefficient and so there is a "piecewise formulation" of being Galois connected. Prove

   $$L \dashv U \quad \equiv \quad L \text{ monotone} \wedge U \text{ monotone} \wedge id \mathrel{\dot\sqsubseteq_C} L \fatsemi U \wedge U \fatsemi L \mathrel{\dot\sqsubseteq_S} id$$

   The right-most two clauses are known as "cancellation laws" since they are similar to the cancellation that occurs for isomorphism —see previous sheet! (We will henceforth dispense with the subscripts.)

2. Specify the notions of top and bottom elements using Galois connections.

3. Define the concept of integer-rounding downwards —the floor function— as a Galois connection. Likewise for integer-rounding upwards.

4. A *pomonoid*, or *p*artially *o*rdered monoid, $(P, \sqsubseteq, \otimes, e)$ is a poset $(P, \sqsubseteq)$ that has a monoid structure $(P, \otimes, e)$ where the two structures interact by the axiom: "the monoid operation is monotone in both arguments":

   $$a \sqsubseteq c \wedge b \sqsubseteq d \quad \Rightarrow \quad a \otimes b \sqsubseteq c \otimes d$$

   Construing the monoid operation $\otimes$ as 'multiplication', we may specify the notion of division '$\oslash$' as a Galois Connection:

   $$x \sqsubseteq n \oslash d \quad \equiv \quad x \otimes d \sqsubseteq n$$

   That is, division '$n \oslash d$' is defined to be the largest number such that when you multiply it back by the denominator d the result still does not exceed the numerator n.

   Prove the following properties,

   (a) We can produce a "direct" definition as $n \oslash m = (\sqcup x \mid x \otimes d \sqsubseteq n \bullet x)$, provided our poset is in-fact a lattice. Moreover, we have: $r \oslash (p \sqcup q) = (r \oslash p) \sqcap (r \oslash q)$ and $(p \sqcap q) \oslash r = (p \oslash r) \sqcap (q \oslash r)$.

   (b) Dividing by the identity is the identity function: $x \oslash e = x$

   (c) Containment as Division: $y \sqsubseteq x \quad \equiv \quad e \sqsubseteq x \oslash y$

   (d) Iterated Division: $(x \oslash z) \oslash y = x \oslash (y \otimes z)$
   Read left-to-right: "Division by a product is the same as successive division by its factors."
   Notice that y and z take turns appearing before the other on each side; why is this?

   Provided our poset is bounded:

   (e) "Dividing by bottom is infinity precisely when bottom is zero": $x \oslash \bot = \top \quad \equiv \quad x \otimes \bot = \bot$
   In particular, if bottom is a zero element, we get the name "dividing by zero is infinity" ;)

   (f) "Infinity is indivisible": $\top \oslash x = \top$

   (g) $n \oslash (m \oslash d) \neq (n \times d) \div m$ —find a counterexample!

   (h) Every lattice gives rise to two pomonoids —see the related quantification exercise above!

   i. Formalise "integer division" as a Galois connection thereby *specifying* the involved notion of integer division —defined in-terms of real division and the floor function— with the simpler notion of integer multiplication. **Prove the division algorithm:** $x \div y =$ **if** $y \leq x$ **then** $(x - y) \div y + 1$ **else** 0 **fi**

   ii. Specify the "weakest precondition" of a program using a Galois Connection.

   Finally, what is division in the conjunctive Boolean pomonoid $(\mathbb{B}, \Rightarrow, \wedge, true)$? Indeed, conjunction has no inverse but has an "approximate inverse" ;) ★

5. **Quantification as Galois Connection**

   Since bounded lattices give us monoids, it is only natural to ask what quantifciations such as $(\sqcup\ x\ \mid\ \mathit{predicate}\ \bullet\ \mathit{body})$ "look like". The quantification $(\sqcup x\ \bullet\ f\ x)$ is also written as $\bigsqcup f$ or as $\mathsf{sup} f$; so let's consider the scenario of specifying the "least upper bound of function f" as a Galois Connection. —then taking f to be the identity function gives a nice characterisation of how to work with arbitrary join- (dually, meet-)quantification! ★

   Essentially, you want to show $\mathsf{sup} \dashv K$ where the "constant function" $K : X \to X \to X$ is the defined by $K\ x\ y = x$, and sup is the operation that takes a function f to its least upper bound:

   $$\text{sup is bound: } f \mathrel{\dot\sqsubseteq} \mathsf{sup}\ f \text{ and least such bound: } \big(\forall u\ \mid\ f \mathrel{\dot\sqsubseteq} K\ u\ \bullet\ \mathsf{sup}\ f \sqsubseteq u\big)$$

   How can this be applied to the *binary* meet and join of a lattice, thereby specfying them as Galois connections? What about the lattice of Boolean values: in `Frama-C` so far you have been using the 'existential quantifier' `\exists` and the 'universal quantifier' `\forall`; expresses these two in the mathematical quantification notation used in this class and use the above result to express them as Galois connections.

6. **Useful Properties of Galois Connections** Needless to say, prove the following properties

   Semi-inverse laws: an adjoint sandwiched by its friend is precisely the friend: $L \fatsemi U \fatsemi L = L$ and $U \fatsemi L \fatsemi U = L$

   Galois connections as quantifications: $U\ y = (\sqcup x\ \mid\ L\ x \sqsubseteq y\ \bullet\ x)$ and $L\ x = (\sqcap x\ \mid\ x \sqsubseteq U\ y\ \bullet\ y)$

   The *L*ower adjoint "can always look up": $L\,(\sqcup x\ \mid\ R\ \bullet\ B) = (\sqcup x\ \mid\ R\ \bullet\ L\ B)$
   The *U*pper adjoint "can always look down": $U\,(\sqcap y\ \mid\ R\ \bullet\ B) = (\sqcap y\ \mid\ R\ \bullet\ U\ B)$

   Hint: **prove** the last two using indirect equality.

7. To conclude this section, show that with 'objects' as posets and 'morphisms' as Galois Connections, we have the structure of a category —see previous sheet!

Exercise 5.4 — Searching: Optional Assignment, 5% —due before the next quiz

1. Choose any **two** searching algorithms of *your choice.*
2. Implement one for lists —the ones you defined in the first exercise above— and implement the other for arrays.
3. Prove your implementation as correct as possible in `Frama-C`

   Part marks will be awarded, as usual.

4. Provide a report of this discovery with the following:
   (a) An abstract indicating the purpose of this report.
   (b) An introduction indicating the intended structure of the report.
   (c) Sections for each implementation including final code-block and intermediary code-blocks, along with colloquial English —your own words!— of how the algorithm works. Afterwards, perhaps in a sub-section, discuss the correctness of the algorithm and provide a code-block with `Frama-C` annotations. Then, in the same section, perhaps in a sub-section, compare and constrast had you used the alternative data structure —arrays vs linked lists.
   (d) A section comparing the two algorithms in-general, comparison them in the linked list approach, and comparing them in the array approach.
   (e) A section including hurdles encountered and conclusion about the report.
5. The following short video may give better guidance to writing a report:
   How to Write a Great Research Paper by Simon Peyton Jones
6. Email your report as a PDF along with the code files to **both** Curtis and myself.

Such a report, if done well, could be used as a selling-factor to impress possible future employers ;)