Homework 4: Abstraction and Free Theorems

15-814: Types and Programming Languages
Fall 2017
Instructor: Karl Crary
TA: Yong Kiam Tan

Out: Oct 16, 2017 (03 pm)
Due: Oct 30, 2017 (11 pm)

Notes:

- Welcome to 15-814's fourth homework assignment!
- Please email your work as a PDF file to `yongkiat@cs.cmu.edu` titled "15-814 Homework 4". Your PDF should be named "<your-name>-hw4-sol.pdf".

1 Abstraction

1.1 Abstraction Theorem

The abstraction theorem (or the parametricity theorem) states that if $e : \tau$, then $e \sim e : \tau$. We can use this theorem to prove several different properties about expressions of polymorphic type.

Task 1 *Show that there are no expressions e of type $\forall\alpha.\forall\beta.\alpha \to \beta$.*

Task 2 *Show that for any expression e of type $\forall\alpha.\alpha \to \alpha \to \alpha$ and any $e_1 : \tau$ and $e_2 : \tau$, either $e[\tau](e_1)(e_2) \sim e_1 : \tau$ or $e[\tau](e_1)(e_2) \sim e_2 : \tau$.*

1.2 Data Abstraction

As emphasized in class, data abstraction is one of the most important concepts in computer science. Let us do an application to get used to this. We will use abstraction to prove that two module implementations are observationally equivalent.

```
signature QUEUE =
sig
        type queue
        val emp : queue
        val ins : int * queue -> queue
        val rem : queue -> unit + (int * queue)
end
```

In this signature,

- emp represents the empty queue.
- ins inserts an element to the back of the queue.

- rem removes the element at the front of the queue and returns it with the remainder of the queue, or `inl`($\star$) if the queue is empty.

Now, consider the following two implementations of this signature.

```
structure LQ : QUEUE =
struct
        type queue = int list
        val emp = [ ]
        fun ins (n, l) = append l [n]
        fun rem l =
                case l of
                [ ] => NONE
                | x::xs => SOME (x, xs)
end

structure LLQ : QUEUE =
struct
        type queue = (int list) * (int list)
        val emp = ([ ], [ ])
        fun ins (n, (front,back)) = (front, n::back)
        fun rem l =
                case (front, back) of
                ([ ], [ ]) => NONE
                | (x::xs,_) => SOME (x, (xs, back))
                | ([ ], _) => rem (rev back, [ ])
end
```

Note that the modules return options NONE and SOME. These can be thought of as syntactic sugar, e.g. think of NONE as `inl`($\star$) and SOME (a, b) as `inr`($\langle$a, b$\rangle$).

In the following tasks, you may assume that the basic types (e.g. `int`, `unit`), and type constructors (e.g. τ `list`, $\tau_1 \times \tau_2, \tau_1 + \tau_2, \tau$ `option`) are already defined. You may also assume that the definition of $\sim$ has been appropriately extended for these types.

Task 3 *State the abstract type of the signature queue.*

Task 4 *State a relation* R : `int list` $\leftrightarrow$ `int list` $\times$ `int list` *which is closed under all operations of the signature.*

Task 5 *Prove that the relation R you defined above is closed under emp.*

Task 6 *Prove that the relation R you defined above is closed under ins.*

Task 7 *Prove that the relation R you defined above is closed under rem.*

Task 8 *Briefly compare the two implementations of queue in terms of time efficiency. Are there situations where you would prefer one over the other?*

2 Free Theorems

As we saw in class, we get some theorems for free from the polymorphic type of an expression

Task 9 *Consider a function* $\mathtt{h} : \forall\alpha.\forall\beta.\alpha\ \mathtt{list} \times \beta\ \mathtt{list} \to (\alpha \times \beta)\ \mathtt{list}$ *(with the type of the standard zip function). Consider functions* $\mathtt{a} : \tau_A \to \tau'_A$ *and* $\mathtt{b} : \tau_B \to \tau'_B$*. Let* $\mathtt{a}^*$ *(similarly for* $\mathtt{b}^*$*) denote the map of* $\mathtt{a}$ *over a list of type* τ_A*, i.e.* $\mathtt{a}^*$ *takes a list* $\mathtt{l}$ *as argument and applies* $\mathtt{a}$ *to every element of* $\mathtt{l}$*. Also, let* $\mathtt{a} \times \mathtt{b} : \tau_A \times \tau_B \to \tau'_A \times \tau'_B$ *be the function where* $(\mathtt{a} \times \mathtt{b})\langle \mathtt{e}_1, \mathtt{e}_2\rangle = \langle \mathtt{a}\ \mathtt{e}_1, \mathtt{b}\ \mathtt{e}_2\rangle$*. Prove that for all* $e : \tau_A\ \mathtt{list} \times \tau_B\ \mathtt{list}$*:*

$$(\mathtt{a} \times \mathtt{b})^*(\mathtt{h}[\tau_A][\tau_B]\ e) \sim \mathtt{h}[\tau'_A][\tau'_B]((\mathtt{a}^* \times \mathtt{b}^*)\ e) : (\tau'_A \times \tau'_B)\ \mathtt{list}$$