GADTs Homework

98-317: Hype for Types

Due: 5 Mar 2019 at 6:30 PM

1 Introduction

In class, we talked about how generalizing ordinary datatypes can allow us to write code that works perfectly and gives us more guarantees at compile time, but also looks really strange. We can write code where branches of a case statement return two different values! In this assignment, you'll look at a couple of pieces of code that involve GADTs (Generalized Algebraic Datatypes) and determine if they typecheck, and if so, what their type is.

Turning in the Homework You should submit a PDF with your solutions to Autolab under GADTs.

2 OCaml

Recall that in class we wrote examples in the language OCaml, the French dialect of ML. OCaml is incredibly similar to SML, with most of the differences being syntactic, but OCaml also has GADTs. The code for the following problems is written in OCaml, so a handy translation guide is provided below:

```
SML:                              OCaml:

datatype 'a foo = Bar | Baz       type 'a foo = Bar | Baz

fn (x : 'a) => x                  fun (x : 'a) -> x

case e of                         match e with
   Bar => ...                        Bar -> ...
 | Baz => ...                      | Baz -> ...
```

3 Types, Types, Types

For the following problems, assume the following two GADTs have been declared.

```
type 'a charm =
   Many : int list -> int charm
 | Single : int -> unit charm
 | Empty : 'a charm
```

```
type 'a strange =
   Hidden : 'b -> int strange
 | Exposed : 'a -> 'a strange
```

In the `strange` type, we've used an *existential type* - one where a type variable appears on the left hand side of the arrow but not the right.

Task 1 Is this expression well typed? If so, give its type. If not, explain why not.

```
fun (type t) (x : t charm) ->
   match x with
      Many _ -> 10
    | Single _ -> ()
    | Empty -> []
```

Task 2 Is this expression well typed? If so, give its type. If not, explain why not.

```
fun (type t) (x : t charm) =>
   match x with
      Many _ -> []
    | Single i -> 10
```

Task 3 Is this expression well typed? If so, give its type. If not, explain why not.

```
fun (x : int strange) ->
   match x with
      Hidden b -> b + 1
    | Exposed i -> i + 1
```

Task 4 Is this expression well typed? If so, give its type. If not, explain why not.

```
fun (x : 'a strange) ->
   match x with
      Exposed i -> i
```