Phantom Types – home assignment

There are five tasks in total. Each task will be graded on a scale from 2 to 5. The overall grade will be the sum of all task grades, divided by four, and rounded down to nearest integer half. The solutions should be presented personally until 13.05.2015. Please contact me via `mkacz91@gmail.com`, facebook (Marcin Kaczmarek), or in person. You are free to choose a programming language and make reasonable modifications.

Task 1. Augment `show` with support of the `Dynamic` type representation, so that

```
show Dynamic (Dyn (Int, 5))
  = "(int = 5)"
show Dynamic (Dyn (Pair (Int, Bool), (5, false)))
  = "(int * bool = (5, false))"
show (List Dynamic) [Dyn (Int, 1); Dyn (Bool, true)]
  = "[ (int = 1) (bool = true) ]"
show Dynamic (Dyn (List Dynamic, [Dyn (Int, 1); Dyn (Bool, true)]))
  = "(dyn list = [ (int = 1) (bool = true) ])"
```

Task 2. The implementation of `cast` presented during the talk performs a deep copy of the casted value. Alter it so that no copying is done.

Task 3. Let's introduce an untyped version of our embedded language.

```
type rawterm =
  | RZero
  | RSucc of rawterm
  | RPred of rawterm
  | RIsZero of rawterm
  | RIf of rawterm * rawterm * rawterm
```

Implement a typechecker that converts an untyped expression into a typed one. A function with signature `rawterm -> 'a term`[1] is not possible without dependent types. A possible solution would be to inject the `term` type into our representations type `typ` and then implement the typechecker as `typecheck : rawterm -> dyn`.

It is then straightforward to introduce `eval_dyn : dyn -> dyn`, such that

```
> let raw = RIf (RIsZero RZero, RSucc RZero, RZero)
val raw : rawterm = RIf (RIsZero RZero, RSucc RZero, RZero)
> let dexpr = typecheck raw
val dexpr : dyn = Dyn (Term Int, <poly>)
> let dval = eval_dyn dexpr
val dval : dyn = Dyn (Int, <poly>)
> show Dynamic dval
- : bytes = "(int = 1)"
```

Task 4. The time consumed by `format` is quadratic with respect to the resulting string[2]. Rewrite it so that only linear time is needed. A solution that accumulates the subsequent parts on a list and then concatenates them using a second pass will not get the maxumum score.

Task 5. Generalize `isum` and `total` to

```
icrush, everything : ('a -> 'a -> 'a -> ) -> 'a -> 'a query -> 'a query
```

so that `icrush ( + ) 0 = isum` and `everything ( + ) 0 = total`.

---

[1] Or rather `rawterm -> 'a term option` since the conversion is not always possible. But since handling of optionals in OCaml is tedious, I chose to indicate failures with exceptions in this task. Hower, you are free to choose how to handle this in your solution.

[2] We assume that concatenating two strings together takes time proportional to the length of the initial one – as if they were lists of characters. This may not be true for some languages and it is indeed not true for OCaml, which has imperative strings, but we ignore that fact for the sake of excercise.