

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science

Final Examination Solutions 2002

Problem 1: Short Answer [29 points]

- a. [4 points] Give a desugaring from a FLAVAR! program P to a FLAVAR! program P' that has the following property: P evaluated with call-by-value semantics has the same behavior as P' when evaluated with call-by-name semantics.

Solution: $\mathcal{D}[(\text{proc } I \ E)] = (\text{proc } I \ \mathcal{D}[(\text{begin } (\text{primop cell-set! } (\text{cell } I) \ I) \ E)])$
For all other E , $\mathcal{D}[E]$ operates by structural induction.

- b. [3 points] Give two domains D and E such that the number of set-theoretic functions from D to E is infinite, but the number of continuous functions is finite.

Solution: $D = \text{integers}$, with \leq as the ordering; $E = \text{bool}$ (unordered)

- c. [3 points] Consider a new construct `smart-compose` to FL/R. (`smart-compose f g`) has the following behavior: it evaluates to whichever of (`compose f g`) or (`compose g f`) type checks. If both type-check, it chooses one of them arbitrarily. Why could type reconstruction using Appendix D fail even when a type for `smart-compose` is reconstructible?

Solution: Type reconstruction could fail if both (`compose f g`) and (`compose g f`) type check (for instance, if f has type $\alpha \rightarrow \beta$ and g has type $\beta \rightarrow \alpha$, for some concrete α and β). An expression that uses (`smart-compose f g`) might require it to have either $\alpha \rightarrow \alpha$ or $\beta \rightarrow \beta$, but type reconstruction might choose the other type. Correcting this problem would require adding backtracking to the algorithm.

- d. [2 points] What is a broken heart, and what is it used for?

Solution: A broken heart is a forwarding pointer, used in copying garbage collection. After an object has been moved, it is replaced by the broken heart, so that references to it can be updated to use the new copy.

- e. [2 points] Give the reconstructed type of the following expression.

```
(lambda (f g)
  (f (f g 2)
     (g 2)))
```

Solution:

```
g: int -> int
f: (int -> int) x int -> (int -> int)
overall: ((int -> int) x int -> (int -> int)) x (int -> int) -> (int -> int)
```


