Assignment #5

Diego Oniarti - 257835

1 Encoding ULC into System F using recursive types

Try to define type τ_u, which is the type that any ULC term can be given in F+isorecursive types. If you can define τ_u, define a function inductively on ULC terms so that it maps any ULC term to a term of F+isorecursive types whose type is τ_u and that has the same behaviour as the original ULC term (i.e., if you map an ULC application, you get something that eventually behaves like an application).

If you cannot define τ_u, argue why it cannot exist.

In this case, consider ULC terms to be: $t ::= n|x|\lambda x.t|t\ t|\langle t,t\rangle|t.1|t.2|\ inl\ t|\ inr\ t|\text{case } t \text{ of } \left| \begin{array}{l} inl\ x_1 \to t \\ inr\ x_2 \to t \end{array} \right.$

Encoding these terms into lambdas is not an option.

No, but why?

2 Formalising ASM capability machines

Capability machines extend assembly instructions with explicit capabilities such that reading and writing on memory is only allowed if a capability is provided.

Take ASM without the private heap and extend the language with capabilities and formalise their semantics: call this CASM. You choose how to model them, choose wisely according to their behaviour as described below.

Capabilities are unforgeable and unobservable tokens which the program can create. At the start, each memory location is unprotected. The language must provide instructions for creating a capability, and for protecting a location given a capability, this should only be possible if the location is unprotected.
Reading and writing a memory location is always possible if the location is unprotected. However, if the location is protected with a capability, reading and writing that location is only possible if the same capability is provided at reading and writing time.

3 Typing CASM

Provide typing rules for all instructions of CASM (that is, all instructions from RML, ASM and CASM).

4 Subtyping

Subtyping lets you use a term of type τ at a super-type τ'. Intuitively, it is the same principle by which you can use an object of a class as if it were of a super class.
Concretely, subtyping is achieved by introducing the subsumption rule below, as well as an ordering on types, indicated with $<:$.

$$\frac{\Gamma \vdash t : \tau \quad \tau <: \tau'}{\Gamma \vdash t : \tau'}\text{Subsumption}$$

Consider the standard typing of the sequencing rule and its related reduction. Suppose we extend our types with $Unit$, so $\tau ::= \cdots | Unit$ and our terms with $unit$ (a new value), so $t ::= \cdots | unit$.

$$\frac{\Gamma \vdash t : Unit \quad \Gamma \vdash t' : \tau}{\Gamma \vdash t; t' : \tau}\text{Type-seq} \qquad \frac{}{unit; t \rightsquigarrow^{p} t}\text{Eval-seq}$$

Consider STLC with sums and pairs. Can we introduce an ordering on types to allow this kind of reduction: $v; t \rightsquigarrow^{p} t$ for any value v, effectively replacing Rule Eval-seq while keeping Rule Type-seq and Rule Subsumption? If yes, show such an ordering. If no, argue why not.