Optimising Compilers

Consider the ML-like language given by abstract syntax

\[ e ::= x \mid \lambda x. e \mid e_1 \ e_2 \mid \text{ref} \ e \mid \text{!} e \mid e_1 := e_2 \mid \text{if} \ e_1 \ \text{then} \ e_2 \ \text{else} \ e_3 \]

where \( x \) ranges over variable names. A previous compiler phase has already determined a type for each variable and has checked the program is well-typed, where types have syntax

\[ \tau ::= \text{int} \mid \text{intref} \mid \tau \rightarrow \tau' \]

Note that there is no polymorphism and moreover if-then-else uses an integer rather than a boolean for a condition and the result of an assignment is the \text{int} value assigned rather than a unit value.

(a) Give an effect system (also known as an annotated type system) in which we can derive judgements of the form

\[ \Gamma \vdash e : t, F \]

where \( t \) is an extended form of \( \tau \) and \( \Gamma \) is a set of assumptions of the form \( x : t \). Effects \( F \) are subsets of \( \{A, R, W\} \) representing that the side-effects of evaluating \( e \) may respectively include a \text{ref} allocation (\( A \)), a dereferencing read (\( R \)) or an update (\( W \)).

(b) Give types and effects for the following programs, commenting briefly on any issues or problems your scheme encounters and how they may be resolved. (Assume \( g \) represents a global variable of type \text{intref}.)

- \( \text{if} \ !g \ \text{then} \ \text{ref} \ 1 \ \text{else} \ g \)
- \( \lambda x. \text{if} \ !g \ \text{then} \ \text{ref} \ x \ \text{else} \ g \)
- \( \text{if} \ !g \ \text{then} \ \lambda x. \text{ref} \ x \ \text{else} \ \lambda x. g \)