A Theory of Type Polymorphism in Programming: The Cheat Sheet

Pete Bevin

April 2015

1 Overview of the Paper

1. Introduction
2. Illustrations of the Type Discipline
3. **A Simple Applicative Language and its Types**
    - The `exp` language, its semantics, and what type errors it can raise at runtime (3.1, 3.2)
    - Types in general (3.3, 3.4)
    - What it means to be well-typed (3.5)
    - Substitutions (3.6)
    - The Semantic Soundness theorem (if we can assign a type, the runtime won't raise errors) (3.7)
4. **A Well-Typing Algorithm and its Correctness**
    - Algorithm W
    - The Syntactic Soundness theorem
    - Algorithm J
5. Types in Extended Languages
    - Tuples, union types, and lists
    - Assignable variables and assignments
    - Recursive type declarations

2 Symbols

Term	Definition in Paper	Meaning
B_0	$T = \{\text{true}, \text{false}, \perp_T\}$	Boolean types (including $\perp$)
$B_1, \ldots, B_n$	Other types	Integers, reals, strings, pairs of types, etc.
V		
W	Wrong	The error type
η	Environment	In the paper, this is a function from `id` to `V`, but we would be more likely to model it as a `map`.
$\mathscr{E}$	Semantic Function	`eval`
$\mathscr{E}[\![T]\!]\eta$	Semantic evaluation of T in environment η	`eval exp env`
ι_n	Type of B_n	For example, B_0 is $\mathbb{B}$, B_1 might be $\mathbb{N}$, B_2 might be $\mathbb{R}$, etc.
$v\textsf{\textbf{E}}D$		`instanceof`
$v \mid D$		Type cast (error value if not possible, but we always check first with $v\textsf{\textbf{E}}D$)
$p \mid e$	Prefixed expression	An expression showing what λ, `fix`, and `let` bindings are in effect
$\overline{p} \mid \overline{e}$	Typed expression	A prefixed expression augmented with type information at each level

3 Concepts

Page	Concept	Meaning
361	Prefix	A list of the variable bindings in effect for an expression
361	Active	A member of the prefix that is not shadowed by a later member with the same name
362	Generic type variable	One that does not occur in the type of any λ or `fix` binding above it
362	Standard typing	In any `let` expression, none of the type variables in `x=e` occur in the `let` body.
362	Well-typed	An expression is well-typed if it can be assigned a type and the resulting type obeys certain laws. See Proposition 3 on page 362.
364	Semantic Soundness Theorem	A well-typed program cannot "go wrong"
367	Syntactic Soundness Theorem	If algorithm $\mathscr{W}$ accepts a program, then it is well-typed.