The Bounded Lattice Type System

Robert Smart (robert.kenneth.smart@gmail.com)

BLTS, the Bounded Lattice Type System, is the key component of the experimental Wombat programming language. BLTS types are characterised by defining and additional properties. For example Integer and Rational might both have "add" properties (perhaps from implementing Behaviour Monoid). Subtyping is initiated with the **isA** declaration which specifies how to do conversions up and down (the latter may fail). Properties which both types have are required to be compatible. Also the lower type acquires any other properties from the higher, so that if Integer **isA** Rational, then Integer acquires the "denominator" property.

The talk will describe the current pre-alpha Wombat implementation which has a number of types and type families, with appropriate **isA** relationships, built in to the interpreter rather than being declared.

There are lattice operations provided by Union and Intersection of any set of types. In particular the Union of the empty set is type Empty, and the Intersection of the empty set is type Any. There are also product types (Tuples), and subset types consisting of a subset of values from a type (with the obvious **isA** relation to the parent). Only subsets with a single value arise in the current implementation.

Optional or repeated computation is done by passing inner closures to appropriate functions (such as while or case). The top level expression in a closure is all executed. The current interpreter does this as follows: Each subexpression starts with type Any, and then gets messages due to identifiers getting set (single assignment), or from the parent or from children. This information drives the type of the expression downwards. As its type changes the expression sends messages to children or parent, or to the controlling code if the node is an identifier that gets set (or at least changed). Eventually the result of running the closure settles to a value (which is a subset type with just one in the subset), or to Empty representing failure. With this type hierarchy directed method of execution, many procedures can be run backwards as well as forwards. Procedures, whether closures or primitive, receive a parameter type and result type, and return improved versions thereof. The result type moves down, but the parameter, if it changes, moves up. For failure the parameter is returned as Any and the result as Empty.

The case procedure (from the **case** operator) moves all the match tests inside the procedures for the match. The specific procedure then fails if it is not a match. The expectation is that only one procedure will succeed with the offered value, and that result is the result for the case. However case has much wider applicability. The Union of procedure types A=>B and C=>D is Intersection(A,C)=>Union(B,D). This is the Wombat case operation. I will show the interpreter code for case that does exactly that, and how it works in some simple examples.