Kalium Pascal Subset

Index Int

February 25, 2015

Kalium supports only a subset of Pascal. It also relies heavily on FPC (Free Pascal Compiler) for error detection—if your program does not compile with FPC, *Kalium* does not promise to detect an error. Instead, it is allowed to generate arbitrary code.

Even if your program compiles, the implemented subset of Pascal is quite limited right now, but will be extended over time.

1 The Implemented Subset

- Data types: `String`, `Char`, `Integer`, `LongInt`, `Real`, `Boolean`. The size and lower/upper bounds of both `Integer` and `LongInt` are the same as those of the `Int` type in Haskell. `Char` is Unicode-aware.
- Basic arithmetics (`+`, `-`, `*`, `/`, `mod`, `div`), logic (`and`, `or`, `xor`, `not`), comparisons (`>`, `<`, `<>`, `>=`, `<=`, `=`).
- Implicit type conversions (`Char` to `String`, `Integer` to `Real`).
- The assignment statement (`:=`), the conditional statement (`if then else`), the for loop (`for to do`), the switch statement (`case of`), statement blocks (`begin end`).
- A variation of the `ReadLn` procedure that accepts exactly one parameter. The `Write` and `WriteLn` procedures.
- User-defined functions and procedures with full input/output support, direct and indirect recursion support, and partial reference parameters support (no aliasing). Local variables.

2 Work in Progress

There is a limited support for dynamic arrays. The `SetLength` procedure is implemented. Indexing (`[]`) is partially implemented: it can be used to access an element of an array or to set it using the assignment statement, but it can't be used to pass an element as a reference parameter. The obvious workaround is to use a temporary variable.

3 The Problem with Aliasing

Though reference parameters in functions and procedures are supported, you must be careful to never introduce aliased variables. In their presence, the semantics of your program may change significantly.

Aliasing happens when different names refer to the same variable. Consider the following function:

```
function f(var a, b: Integer): Integer;
begin
    a := 0;
    f := b;
end.
```

It changes the value of `a` and returns `b` unchanged. But if you call it as `f(x,x)`, then inside the function both `a` and `b` refer to `x`, so changing the value of `a` changes the value of `b`! As of yet, *Kalium* does not handle this behaviour properly. The rule of thumb to avoid aliasing: don't ever pass the same variable as different reference parameters.

The solution to this problem would be to generate multiple versions of a function, one for every possible combination of aliased variables. Unfortunately, it also would lead to massive code duplication.