Design of YASS

Ichinose Kaori

December 20, 2023

1 Deviations from the R^{7}RS-small Standard

- In section 4.1, inclusion is removed as a primitive expression type.
- Macros are Common-Lisp-style.
- In section 5.3, variable definitions are constrained to the first form.
- In section 6.2, YASS implements only operations on signed 61-bit integers and 64-bit floating point numbers, and conversions between them.

2 Pass Organization

YASS is a micropass compiler with a similar structure to SML/NJ. Its hierarchy of IRs are:

Source language Scheme with deviations from the standard described in section 1.

Primitives ** after macro expansion. Contains variable references, quotes, applications, abstractions, conditionals, and assignments.

AST ** in AST form so that calls to primitives generated in later passes would not shadow user program variables.

Unique-names ** where local names are unique.

Known-adic ** where there are no variadic functions. Abstractions no longer have a rest argument. `Cons` is added as a primitive.

Assign-less ** where there are no assignments. `Make-box`, `box-ref`, and `box-set!` are added as primitives.

CPS CPS with added primitives.

ClosurePS ** where functions are closed. YASS uses flat closures. `Make-closure`, `call-closure`, and `closure-ref` are added as primitives.

Tag bits	Meaning
`000`	61-bit signed integer
`001`	character
`010`	symbol
`011`	special values (null, boolean, errors, etc.)
`100`	pointer to heap
`101`	unused
`110`	unused
`111`	unused

Table 1: Meanings of tag bits

VM ** where names are replaced with indexed accesses. `Argument-ref` is added as a primitive.

Registered ** where the number of arguments of functions are bounded.

Machine language Machine language. This is usually a very small subset of the target.

3 Runtime Specifics

Runtime of YASS runs on top of the C runtime. This enables YASS programs to call the operating system or to use C libraries through a C FFI, and the runtime to be portable.

4 Datum Representation

Data in YASS are represented as 64-bit tagged pointers, with the 3 MSBs for the tag. The meanings of the tag bits are shown in table 1.

Objects on the heap are aligned to the 64-bit barrier and the formats for storing them are as follows:

Bytevector A pointer with tag 0 and value the length at position 0 and elements aligned to an 8-byte barrier follow.

Pair A pointer with tag 1 at position 0 and car and cdr follow.

Port A pointer with tag 2 and value the `FILE *` at position 0.

Procedure A pointer with tag 3 and value the length of the procedure specification at position 0 and the machine code follows.

String A pointer with tag 4 and value the length of the string at position 0 and elements follow.

Vector Similar to strings but with a tag of 5.

Floating point number A pointer with tag 6 at position 0 and the floating point number representation follows.

Record A pointer with tag 7 and value type of the record, then a number representing the number of fields, then fields.

Environment A special record type. Contains an associative map of syntactic keywords to syntax transformers, and an associative map from top-level variable names to values.