Design of YASS

Ichinose Kaori

January 20, 2024

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. Numeric operations are not polymorphic.
- Datum labels are removed.
- Libraries are removed.
- `Begin` is no longer used for splicing definitions into a context, except at the top-level.

2 Core Grammar

This section describes the core grammar of the language implemented by YASS.

⟨*program*⟩ ::= ⟨*definition*⟩ ⟨*program*⟩
  | empty

⟨*definition*⟩ ::= (define ⟨*identifier*⟩ ⟨*expr*⟩)
  | (define-syntax ⟨*identifier*⟩ ⟨*expr*⟩)

⟨*expr*⟩ ::= ⟨*identifier*⟩
  | (quote ⟨*datum*⟩)
  | (if ⟨*expr*⟩ ⟨*expr*⟩)
  | (if ⟨*expr*⟩ ⟨*expr*⟩ ⟨*expr*⟩)
  | (lambda ⟨*formals*⟩ ⟨*expr*⟩)
  | ⟨*application*⟩

⟨*application*⟩ ::= (⟨*expr*⟩)
  | ⟨*expr*⟩ :: ⟨*application*⟩

⟨*formals*⟩ ::= ⟨*identifier*⟩
| ()
| ⟨*identifier*⟩ :: ⟨*formals*⟩

3 Pass Organization

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

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

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

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.

CPS CPS with added primitives.

Known-adic ** where variadic functions are eliminated. This is done by converting functions so that they take a single list as an argument and deconstruct the list in the function body. `Vector`, `vector->list`, and `vector-ref` are added as primitives.

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

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 a very small subset of the target.

4 Runtime Specifics

The runtime 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.

Garbage collection is done by checking for a GC cycle before each CPSed function.

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

Value	Meaning
0	false
1	true
2	null

Table 2: Special values

5 Datum Representation

Data in YASS are represented as 64-bit tagged pointers, with the 3 LSBs 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:

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

Port A pointer with tag 1 and value the `FILE *` at position 0.

Procedure A pointer with tag 0 and value 1 at position 0 and a pointer to machine code and vector to enclosed values follows.

String A pointer with tag 2 and value the length of the string at position 0 and elements follow.

Vector Similar to strings but with a tag of 3.

Floating point number A pointer with tag 0 at position 0 and value 2 and the floating point number representation follows.

Reference cell A pointer with tag 0 at position 0 and value 3 and the boxed pointer follows.

Tombstone A pointer with tag 0 at position 0 and value 4, and the forward pointer follows.