Macrolop Specification

Temirkhan Myrzamadi (a.k.a. Hirrolot)

November 18, 2020

Abstract

This paper is the official specification of Macrolop, a metalanguage aimed at language-oriented programming in C. In it, both the syntax and reduction semantics are defined formally. See the official repository [1] for the user-friendly overview and the official documentation [2] for the accompanied standard library.

Contents

1 EBNF Grammar

```
<eval> ::= "MACROLOP_EVAL(" { <term> }* ")" ;

<term> ::= "call(" <op> "," { <term> }* ")"
         | "v(" <preprocessor-token-list> ")" ;

<op>   ::= <ident> | { <term> }+ ;
```

Figure 1: Grammar rules

A metaprogram in Macrolop consists of a (possibly empty) sequence of terms, each of which is either a macro call or just a value.

Notes:

- The grammar above describes metaprograms already expanded by the C preprocessor, except for `MACROLOP_EVAL`, `call`, and `v`.
- `call` accepts `op` either as an identifier or as a non-empty sequence of terms that reduces to an identifier.
- `call` accepts arguments without a separator.

2 Reduction Semantics

We define reduction semantics for Macrolop. The abstract machine executes configurations of the form $\langle k; acc; control \rangle$:

- k is a continuation of the form $\langle k; acc; control \rangle$, where *control* include the ? sign, which will be substituted with a result after a continuation is called. For example: let $k = \langle k'; (1, 2, 3); v(abc)\ ? \rangle$, then $k(v(ghi))$ is $\langle k'; (1, 2, 3); v(abc)\ v(ghi) \rangle$. A special continuation *halt* terminates the abstract machine with provided result.
- *acc* is an accumulator, a sequence 2 of already computed results.
- *control* is a concrete sequence of terms upon which the abstract machine is operating right now. For example: `call(FOO, v(123) v(456)) v(w 8) v(blah)`.

And here are the computational rules:

$$\begin{array}{rll}
(v): & \langle k; acc; v(\overline{tok})\ t\ \overline{t'}\rangle & \rightarrow_1 \langle k; acc,\ \overline{tok}; t\ \overline{t'}\rangle \\
(v\text{-}end): & \langle k; acc; v(\overline{tok})\rangle & \rightarrow_1 k(unseq(acc, \overline{tok})) \\
(op): & \langle k; acc; call(\overline{t}, \overline{a})\ \overline{t'}\rangle & \rightarrow_1 \langle\langle k; acc; call(?, \overline{a})\ \overline{t'}\rangle; (); \overline{t}\rangle \\
(args): & \langle k; acc; call(ident, \overline{a})\ \overline{t}\rangle & \rightarrow_1 \langle\langle k; acc; ident(unseq\text{-}cs(?))\ \overline{t}\rangle; (); \overline{a}\rangle \\
(start): & MACROLOP\_EVAL(\overline{t}) & \rightarrow_1 \langle halt; (); \overline{t}\rangle
\end{array}$$

Figure 2: Computational rules

Notation 1 (Sequences)

1. *A sequence has the form* $(x_1, \ldots, x_n)$.
2. $()$ *denotes the empty sequence.*
3. *An element can be appended by comma: if* $a = (1, 2, 3)$ *and* $b = 4$*, then* $a, b = (1, 2, 3, 4)$.
4. `unseq` *extracts elements from a sequence without a separator:* `unseq( (a, b, c)) = a b c`.
5. `unseq-cs` *extracts elements from a sequence separated by comma:* `unseq-cs((a, b, c)) = a, b, c`.

Notation 2 (Reduction step)
$\rightarrow_1$ *denotes a single step of reduction (computation).*

Notation 3 (Concrete sequence)
$\overline{x}$ *denotes a concrete sequence* $x_1 \ldots x_n$. *For example:* `v(abc) call(FOO, v(123)) v(u 8 9)`.

Notation 4 (Meta-variables)

`tok`	*C preprocessor token*
`ident`	*C preprocessor identifier*
`t`	*Macrolop term*
`a`	*Macrolop term used as an argument*

Notes:

- A body of a macro called using `call` must follow the grammar of Macrolop, otherwise it might result in a compilation error.
- With the current implementation, at most 2^{14} reduction steps is possible. After exceeding this limit, compilation will likely fail.

References

[1] Temirkhan Myrzamadi. *Language-oriented programming in C.* URL: `https://github.com/Hirrolot/macrolop`.

[2] Temirkhan Myrzamadi. *The Macrolop standard library documentation.* URL: `https://hirrolot.github.io/macrolop/`.