# Macrolop Specification

Temirkhan Myrzamadi (a.k.a. Hirrolot)

November 12, 2020

## 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.

Note that a macro call accepts arguments without a separator because otherwise there must be logic to avoid putting a comma after the last argument (if they're generated programmatically). However, this design decision tends to break IDE support (code formatting, macro parameters hightlighting, ...). The common workaround is to write a `*_REAL` macro (the actual implementation) and a C-style wrapper:

```
#define FOO(a, b, c) FOO_REAL(a b c)
#define FOO_REAL(a, b, c) // Implementation...
```

Then `FOO` can be called as `FOO(a, b, c)`, where `a`, `b`, and `c` stand for the actual arguments.

Note that the given syntax holds for metaprograms already expanded by the C preprocessor, except for the macros `MACROLOP_EVAL`, `call`, and `v`. So a syntactically well-formed metaprogram in Macrolop is a C metaprogram that expands to a sequence of preprocessor tokens (again except for the aforementioned cases) matching the given grammar.

Also note that `call` accepts `op` either as an identifier or as a term that computes to an identifier. For instance, you can write both `call(FOO, ...)`, `call(v(FOO), ...)`, and even `call(call(BAR, ...), ...)` as long as `call(BAR, ...)` reduces to an identifier.

# 2 Operational Semantics

We define small-step operational semantics for Macrolop. Take into consideration the following notations:

| | |
|---|---|
| $\to_1$ | a single step of computation |
| $\mathit{term}$ | `<term>` |
| $a$ | `<term>` used as a macro argument |
| $\mathit{tok}$ | `<preprocessor-token>` |
| $x\ldots$ | a possibly empty sequence $x_1, \ldots, x_n$ |
| $\mathit{empty}$ | an empty sequence |
| $\langle \mathit{acc}; x\rangle$ | $x$ with the accumulator $\mathit{acc}$ |
| $\mathit{op}(...)$ | a C-style macro call |
| $\mathit{ident}$ | a C identifier (`foo`, `bar`, ...) |

$$(\text{v})\ \frac{}{\langle \sigma; v(\mathit{tok}\ldots)\ \mathit{term}\ldots\rangle \to_1 \langle \sigma\ \mathit{tok}\ldots; \mathit{term}\ldots\rangle}$$

$$(\text{call})\ \frac{\mathit{call}(\mathit{op}, a\ldots) \to_1 \mathit{term}\ldots}{\langle \sigma; \mathit{call}(\mathit{op}, a\ldots)\ \mathit{term}'\ldots\rangle \to_1 \langle \sigma; \mathit{term}\ldots\ \ \mathit{term}'\ldots\rangle}$$

$$(\text{eval-op-step})\ \frac{\mathit{term} \to_1 \mathit{term}'}{\mathit{call}(\mathit{term}\ldots, a\ldots) \to_1 \mathit{call}(\mathit{term}'\ldots, a\ldots)}$$

$$(\text{eval-op})\ \frac{\mathit{term} \to_1 \mathit{ident}}{\mathit{call}(\mathit{term}, \mathit{term}'\ldots) \to_1 \mathit{call}(\mathit{ident}, \mathit{term}'\ldots)}$$

$$(\text{arg-call})\ \frac{\langle \sigma; \mathit{term}\ldots\rangle \to_1 \langle \sigma\ \mathit{tok}\ldots; \mathit{term}'\mathit{term}''\ldots\rangle}{\langle \sigma; \mathit{call}(\mathit{ident}, \mathit{term}\ldots)\rangle \to_1 \langle \sigma\ \mathit{tok}\ldots; \mathit{call}(\mathit{ident}, \mathit{term}'\mathit{term}''\ldots)\rangle}$$

$$(\text{arg-call})\ \frac{\langle \sigma; \mathit{term}\rangle \to_1 \langle \sigma\ \mathit{tok}\ldots; \mathit{empty}\rangle}{\langle \sigma; \mathit{call}(\mathit{ident}, \mathit{term})\rangle \to_1 \langle \mathit{empty}; \mathit{ident}(\sigma\ \mathit{tok}\ldots)\rangle}$$

**Figure 2:** Computational rules

Note that a body of a macro called using `call` must follow the grammar of Macrolop, otherwise it might result in a compilation error.