Metamath C technical appendix

MARIO CARNEIRO, Carnegie Mellon University

1 INTRODUCTION

This is an informal development of the theory behind the Metamath C language: the syntax and separation logic, as well as the lowering map to x86. For now, this is just a set of notes for the actual compiler. (Informal is a relative word, of course, and this is quite formally precise from a mathematician's point of view. But it is not mechanized.)

2 SYNTAX

The syntax of MMC programs, after type inference, is given by the following (incomplete) grammar:

$$
\begin{array}{rcll}
\alpha, x, h \in \mathrm{Ident} & ::= & \text{identifiers} & \\
s \in \mathrm{Size} & ::= & 8 \mid 16 \mid 32 \mid 64 \mid \infty & \text{integer bit size} \\
t \in \mathrm{TuplePattern} & ::= & x \mid \boxed{x} & \text{variable, ghost variable} \\
& \mid & t : \tau \mid \langle \overline{t} \rangle & \text{type ascription, tuple} \\
\tau \in \mathrm{Type} & ::= & \alpha & \text{type variable reference} \\
& \mid & \mathbf{0} \mid \mathbf{1} \mid \mathsf{bool} & \text{void, unit, booleans} \\
& \mid & \mathbb{N}_s \mid \mathbb{Z}_s & \text{unsigned and signed integers of different sizes} \\
& \mid & \tau[pe] & \text{arrays of known length} \\
& \mid & \mathsf{own}\ \tau \mid \&\tau \mid \&^{\mathbf{mut}}\tau & \text{owned, borrowed, mutable pointers} \\
& \mid & \bigcap \overline{\tau} \mid \bigcup \overline{\tau} & \text{intersection type, (undiscriminated) union type} \\
& \mid & \bigast \overline{\tau} \mid \sum \overline{R} & \text{tuple type, structure (dependent tuple) type} \\
& \mid & S(\overline{\tau}, \overline{pe}) & \text{user-defined type} \\
& \mid & \ldots & \\
A \in \mathrm{Prop} & ::= & e & \text{assert that a boolean value is true} \\
& \mid & \top \mid \bot \mid \mathsf{emp} & \text{true, false, empty heap} \\
& \mid & \forall x : \tau,\ A \mid \exists x : \tau,\ A & \text{universal, existential quantification} \\
& \mid & A_1 \to A_2 \mid \neg A & \text{implication, negation} \\
& \mid & A_1 \wedge A_2 \mid A_1 \vee A_2 & \text{conjunction, disjunction} \\
& \mid & A_1 * A_2 \mid A_1 \mathrel{-\!\!*} A_2 & \text{separating conjunction and implication} \\
& \mid & \ldots & \\
R \in \mathrm{Arg} & ::= & x : \tau \mid \boxed{x} : \tau \mid h : A & \text{regular/ghost/proof argument}
\end{array}
$$

Author's address: Mario Carneiro, Carnegie Mellon University.

$$
\begin{array}{rcll}
pe \in \mathrm{PExpr} & ::= & \text{(the first half of Expr below)} & \text{pure expressions} \\
e \in \mathrm{Expr} & ::= & x & \text{variable reference} \\
& | & e_1 \wedge e_2 \mid e_1 \vee e_2 \mid \neg e & \text{logical AND, OR, NOT} \\
& | & e_1 \mathbin{\&} e_2 \mid e_1 \mathbin{|} e_2 \mid !_s\ e & \text{bitwise AND, OR, NOT} \\
& | & e_1 + e_2 \mid e_1 * e_2 \mid -e & \text{addition, multiplication, negation} \\
& | & e_1 < e_2 \mid e_1 \le e_2 \mid e_1 = e_2 & \text{equalities and inequalities} \\
& | & \mathsf{if}\ h^? : e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 & \text{conditionals} \\
& | & \langle \overline{e} \rangle & \text{tuple} \\
& | & f(\overline{e}) & \text{(pure) function call} \\
& | & \mathsf{let}\ h^? := t := e_1\ \mathsf{in}\ e_2 & \text{assignment to a regular variable} \\
& | & \mathsf{let}\ t := p\ \mathsf{in}\ e & \text{assignment to a hypothesis} \\
& | & \mathsf{mut}\ x\ \mathsf{in}\ e & \text{mutation capture} \\
& | & F(\overline{e}) & \text{procedure call} \\
& | & \mathsf{unreachable}\ p & \text{unreachable statement} \\
& | & \mathsf{return}\ \overline{e} & \text{procedure return} \\
& | & \mathsf{let\ rec}\ \overline{\ell(\overline{x}) := e}\ \mathsf{in}\ e & \text{local mutual tail recursion} \\
& | & \mathsf{goto}\ \ell(\overline{e}) & \text{local tail call} \\
& | & \ldots & \\
p \in \mathrm{Proof} & ::= & \mathsf{entail}\ \overline{p}\ q & \text{entailment proof} \\
& | & \mathsf{assert}\ pe & \text{assertion} \\
& | & \ldots & \\
q \in \mathrm{RawProof} & ::= & \ldots & \text{MM0 proofs} \\
it \in \mathrm{Item} & ::= & \mathsf{type}\ S(\overline{\alpha}, \overline{R}) := \tau & \text{type declaration} \\
& | & \mathsf{const}\ t := e & \text{constant declaration} \\
& | & \mathsf{global}\ t := e^? & \text{global variable declaration} \\
& | & \mathsf{func}\ f(\overline{R}) : \overline{R} := e & \text{function declaration} \\
& | & \mathsf{proc}\ f(\overline{R}) : \overline{R} := e & \text{procedure declaration}
\end{array}
$$

Missing elements of the grammar include:

- Switch statements, which are desugared to if statements.
- Raw MM0 formulas can be lifted to the ‘Prop’ type.
- Raw MM0 values can be lifted into $\mathbb{N}_\infty$ and $\mathbb{Z}_\infty$.
- There are more operations for indexing and slicing array references, as well as assigning to parts of an array.