

Realizzabilità e altre cose interessanti

Alberto Fiori

ABSTRACT. As I understand it the speciality of a realizability interpretation is in being looser, for example in this theory the term $natrec(x, 0, 0)$, where x is a free variable, belongs to every type despite not even being well-formed in a type theory context.

1. Copycatting: Let's get vaguely serious

1.1. Starting definitions. We are going to give a different interpretation of the same language used in the Minimalist Foundation. Una delle differenze sarà che definiremo un'untyped relazione di riducibilità invece del giudizio di uguaglianza tipata; then using the distinction in **canonical** and **non-canonical** expression we will define the set of **normal** expressions (le espressioni normali in mtt sono normali anche qui)

CHAPTER 1

Expression Language

The following is based on the ideas that I first read in the book "programming in Martin L f type theory" ¹. Basically every expression we write in a mathematical expression has an arity $(0, \alpha \rightarrow \beta, \alpha_1 \otimes \alpha_2 \dots \otimes \alpha_n)$ which work similarly to how types in simply typed lambda calculus work. This simply is a way to ensure that expression are well-formed (even if it doesn't ensure that they are reasonable); we chose to add the $\otimes \dots \otimes$ constructor to freely chose when to *curry* and *uncurry* function application; we could have added more, but there seem to be no advantage in doing so.

The rule for arity are the expected ones:

1. Everithhung has an Arity

123123123123

¹Non sono pi  convinto di questa cosa; nel libro si parla esclusivamente di *MLTT* dove c'  un enorme rigidit  nelle costruzioni ammesse, al contrario quando siamo in realizzabilit  abbiamo decisamente una maggior libert : potremmo per esempio scrivere l'espressione $\text{apply}(0,0)$ semplicemente questa non farebbe parte di alcun tipo. rimarrebbe ora da decidere come trattare il concetto di ariet  funzionale, la Coquand definisce il costruttore di funzione come $\lambda x.e$ dando nomi espliciti alle variabili e richiamandosi all'operatore di sostituzione $B\{a/x\}$ per rappresentare i "conti" seguendo questa struttura, la consueta sintassi di funzione diventa obsoleta, il lambda calcolo funzionale (i.e. $(x).f$ o $\langle x \rangle.f$) diventa di fatto non strettamente necessario negli usi che servono nell'articolo.

Questo mi porta a propormi di non accanirmi sulle ariet  di non preoccuparsi troppo del $\lambda x.e$ dei Π -tipi (a cui non sono troppo abituato: $\lambda x.e = \lambda((x).e)$)

2. The previous section was a bad idea

As already pointed out in the footnote 1 on page 3 it was not a good idea to start from the wellformedness of formulas as to prove the normalization theorem we need just the concepts of **canonical** and **non-canonical** and the **tree-like** structure of expressions to define the reducibility relation recursively.

In the whole article, I think, nothing else was used.

CHAPTER 2

Test stuff

$$(1) \quad \frac{A \quad (A \rightarrow B)}{B}$$

$$(2) \quad \frac{A \quad (A \rightarrow B)}{B}$$

$$(3) \quad \frac{A \wedge B}{B \wedge A}$$

$$(4) \quad \frac{(\neg\phi \vee \neg\psi) \quad \perp \quad \perp}{\perp} \quad (1)$$

$$(5) \quad \frac{\perp}{\neg(\phi \wedge \psi)} \quad (2)$$

$$(6) \quad \frac{(\neg\phi \vee \neg\psi) \quad \frac{\phi \quad \neg\phi}{\perp} \quad \perp}{\perp} \quad (1)$$

$$(7) \quad \frac{\perp}{\neg(\phi \wedge \psi)} \quad (2)$$

$$\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array}}{(A \rightarrow B)} (\rightarrow I) \quad \frac{(A \rightarrow B) \quad A}{B} (\rightarrow E)$$

The rule $\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array}}{(A \rightarrow B)}$ is known as \rightarrow -intro

$$\frac{\frac{\vdash A \quad \frac{\vdash B}{\vdash B, C}}{\vdash A \wedge B, C}}{\vdash A \wedge B, C} \rightsquigarrow \frac{\frac{\vdash A \quad \vdash B}{\vdash A \wedge B}}{\vdash A \wedge B, C}$$