CHAPTER 1

Expression Language

The following is based on the ideas that I first read in the book "programming in Martin Löf type thery" [1]. Basically every expression we write in a mathematical expression has an arity ($0, \alpha \to \beta, \alpha_1 \otimes \alpha_2 \ldots \otimes \alpha_n$) 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 $\otimes \ldots \otimes$ 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

[1]Non sono più convinto di questa cosa; nel libro si parla escusivamente di *MLTT* dove c'è un enorme rigidità nelle costruzioni ammesse, al contrario quando siamo in realizzabilità abbiamo decisamente una maggior libertà: potremmo per esempio scrivere l'espressione apply(0,0) semplicemente questa non farebbe parte di alcun tipo. rimarrebbe ora da decidere come trattare il concetto di arietà 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 a rietà di non preoccuparsi troppo del $\lambda x.e$ dei Π-tipi (a cui n on 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 1 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 ricursively.

CHAPTER 2

Test stuff

$$\frac{A \quad (A \to B)}{B} \tag{1}$$

$$\frac{A \quad (A \to B)}{B} \tag{2}$$

$$\genfrac{}{}{1.5pt}{}{A \wedge B)}{B \wedge A} \tag{3}$$

$$\dfrac{\dfrac{(\neg\phi \vee \neg\psi) \quad \bot \quad \bot}{\bot}\ (1)}{\neg(\phi \wedge \psi)}\ (2) \tag{4}$$

$$\dfrac{\dfrac{(\neg\phi \vee \neg\psi) \quad \dfrac{\phi \quad \overline{\neg\phi}^{\,(1)}}{\bot} \quad \bot}{\bot}\ (1)}{\neg(\phi \wedge \psi)}\ (2) \tag{5}$$

(6)

$$\dfrac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{(A \to B)}\ (\to I) \qquad \dfrac{(A \to B) \quad A}{B}\ (\to E)$$

The rule $\dfrac{\begin{array}{c}[A]\,A \quad A \quad A \\ \vdots \\ B\end{array}}{(A \to B)}$ is known as $\to$-intro

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