# Mathematical Foundations

David McAllester

January 2010

## 1 Getting Started

Our objective is to formalize the notion of a well formed mathematical expression. Here we define a first system which will define a set of well formed expressions for arithmetic including the ability to define arithmetic functions and predicates in a flexible way.

A fundamental kind of mathematical expression has the form $f(x_1, \ldots, x_n)$ where $f$ is a function symbol and $x_1$, ..., $x_n$ are variables. For this to be well formed we require that the variables have types which are compatible with the types of the function $f$. We write the type of $f$ as follows.

$$f : \tau_1 \times \cdots \times \tau_n \to \sigma$$

Here $\tau_1$, ..., $\tau_n$ and $\sigma$ are all types. In this section we consider only the three types **Boole**, and **int**, Since we are doing mathematics rather than programming we do not have to worry about floating point numbers or precision and we can talk about actual (infinite precision) real numbers such as $\pi$.

The set of well formed expressions is defined by type inference rules. Our first type inference rule is the rule for typing applications.

$$\text{A1}\qquad \frac{\begin{array}{l}\Sigma \vdash e_1 : \tau_1 \\ \vdots \\ \Sigma \vdash e_n : \tau_n \\ \Sigma \vdash f : \tau_1 \times \cdots \times \tau_n \to \sigma\end{array}}{\Sigma \vdash f(e_1, \ldots, e_n) : \sigma}$$

In general the expression $e : \tau$ states that $e$ is a well formed expression of type $\tau$. The expression $\Sigma \vdash \Phi$ is called a sequent and expresses the statement that the assumptions in $\Sigma$ allow one to derive the statement $\Phi$. For now we assume that we can derive the following.

$$f : \mathbf{int} \to \mathbf{int}\ ;\ x : \mathbf{int} \vdash x : \mathbf{int} \tag{1}$$

$$f : \mathbf{int} \to \mathbf{int} \; ; \; x : \mathbf{int} \vdash f : \mathbf{int} \to \mathbf{int} \tag{2}$$

From (1) and (2) and the axiom A1 we can derive the following sequent.

$$f : \mathbf{int} \to \mathbf{int}, \; x : \mathbf{int} \quad \vdash \quad f(x) : \mathbf{int} \tag{3}$$

This sequent says that if $f$ is a function from integers to integers and x is an integer then $f(x)$ is also an integer. By another use of the axiom A1 we can derive the following.

$$f : \mathbf{int} \to \mathbf{int}, \; x : \mathbf{int} \quad \vdash \quad f(f(x)) : \mathbf{int} \tag{4}$$

We will also be able to derive the following.

$$g : \mathbf{int} \times \mathbf{int} \to \mathbf{int}, \; x_1 : \mathbf{int}, \; x_2 : \mathbf{int} \quad \vdash \quad g(x_1, x_2) : \mathbf{int} \tag{5}$$

While the sequents (1) and (2) may seem trivial we want to restrict the derivation of sequents of this form to cases where the assumption set $\Sigma$ is itself well formed. For this reason we require that sequents such as (1) and (2) be derivable from the following structural rules.

A2
$$\frac{}{\emptyset \vdash \mathbf{int} : \mathbf{type}}$$

A3
$$\frac{\begin{array}{l}\Sigma \vdash \tau_1 : \mathbf{type} \\ \vdots \\ \Sigma \vdash \tau_n : \mathbf{type} \\ \Sigma \vdash \sigma : \mathbf{type}\end{array}}{\Sigma \vdash (\tau_1 \times \cdots \times \tau_n \to \sigma) : \mathbf{type}}$$

A4
$$\frac{\begin{array}{l}\Sigma \vdash \tau : \mathbf{type} \\ x \text{ is a variable} \\ x \text{ does not occur } \Sigma \\ x \text{ does not occur in any foundational axiom}\end{array}}{\Sigma \; ; \; x : \tau \vdash x : \tau}$$

A5
$$\frac{\begin{array}{l}\Sigma \vdash \Phi \\ \Sigma \; ; \; \Theta \vdash \Psi\end{array}}{\Sigma \; ; \; \Theta \vdash \Phi}$$

We now have five axioms numbered A1 through A5. We will continue to number axioms as they are added to our system. The axioms A2 through A5 require that we use well-formed assumption sets. Each type expression in a type declaration must itself be well formed and a give variable can only have a single type declared for it.

Here we write $\emptyset$ to denote the empty assumption set. Axiom A2 allows us to derive the following.

$$\emptyset \vdash \mathbf{int} : \mathbf{type} \tag{6}$$

Axiom A3 then can be used to derive the following.

$$\emptyset \vdash (\mathbf{int} \to \mathbf{int}) : \mathbf{type} \tag{7}$$

Axiom A4 then can be used to derive the following.

$$\emptyset \ ; \ f : \mathbf{int} \to \mathbf{int} \vdash f : \mathbf{int} \to \mathbf{int} \tag{8}$$

Axiom A5 then lets us derive the following.

$$\emptyset \ ; \ f : \mathbf{int} \to \mathbf{int} \vdash \mathbf{int} : \mathbf{type} \tag{9}$$

Axiom A4 then lets us derive the following.

$$\emptyset \ ; \ f : \mathbf{int} \to \mathbf{int} \ ; \ x : \mathbf{int} \vdash x : \mathbf{int} \tag{10}$$

Axiom A5 then lets us derive the following.

$$\emptyset \ ; \ f : \mathbf{int} \to \mathbf{int} \ ; \ x : \mathbf{int} \vdash f : \mathbf{int} \to \mathbf{int} \tag{11}$$

Here (10) and (11) are slight variations of (1) and (2) respectively where the we have added the empty assumption set $\emptyset$ to the beginning of the assumptions. The sequents (3), (4) and (5) should be similarly modified. We will generally not write $\emptyset$ at the beginning of our assumption sets but all assumption sets should technically start with $\emptyset$.

We also add axioms declaring types for the following foundational constants associated with the integers.

A6 $\dfrac{}{\emptyset \vdash 0 : \mathbf{int}}$

A7 $\dfrac{}{\emptyset \vdash 1 : \mathbf{int}}$

A8 $\dfrac{}{\emptyset \vdash -1 : \mathbf{int}}$

A9 $\dfrac{}{\emptyset \vdash + : \mathbf{int} \times \mathbf{int} \to \mathbf{int}}$

A10 $\dfrac{}{\emptyset \vdash * : \mathbf{int} \times \mathbf{int} \to \mathbf{int}}$

We will generally write applications of $+$ and $*$ using infix notation. In other words we will write $x + y$ rather than $+(x, y)$.

## 2 Boolean Expressions

We have the following foundational boolean connectives where $\vee$ is "or", $\wedge$ is "and", $\Rightarrow$ is "implies", and $\neg$ is "not". As with the foundational integer constants, these Boolean connectives can always be assigned the associated type.

A11 $$\frac{}{\emptyset \vdash \mathbf{Boole} : \mathbf{type}}$$

A12 $$\frac{}{\emptyset \vdash \vee : \mathbf{Boole} \times \mathbf{Boole} \to \mathbf{Boole}}$$

A13 $$\frac{}{\emptyset \vdash \wedge : \mathbf{Boole} \times \mathbf{Boole} \to \mathbf{Boole}}$$

A14 $$\frac{}{\emptyset \vdash \ \Rightarrow : \mathbf{Boole} \times \mathbf{Boole} \to \mathbf{Boole}}$$

A15 $$\frac{}{\emptyset \vdash \neg : \mathbf{Boole} \to \mathbf{Boole}}$$

We allow for equality formulas between objects of the same type. In particular we allow the following type inference rule for equality.

A16 $$\frac{\Sigma \vdash u : \tau \qquad \Sigma \vdash v : \tau}{\Sigma \vdash u = v : \mathbf{Boole}}$$

It is very important that equality can only be applied between expressions that can be assigned the same type (we will see later that the same expression can sometimes be assigned several types). So we can ask if two integers are equal. But it is ill-formed to ask if a formula (an expression of type **Boole**) is equal to the integer zero.

We can also form quantified formulas using the "for all" quantifier $\forall$ and the "there exists" quantifier $\exists$. In particular we have the following rules.

$$\text{A17}\qquad \frac{\Sigma; x : \tau \vdash \Phi : \mathbf{Boole}}{\Sigma \vdash (\forall x : \tau\ \Phi) : \mathbf{Boole}}$$

$$\text{A18}\qquad \frac{\Sigma; x : \tau \vdash \Phi : \mathbf{Boole}}{\Sigma \vdash (\exists x : \tau\ \Phi) : \mathbf{Boole}}$$

We can now derive the following.

$$x : \mathbf{int},\ y : \mathbf{int} \quad \vdash \quad x = y\ :\ \mathbf{Boole}$$

By two applications of A17 we can then derive the following.

$$\vdash\ (\forall x : \mathbf{int}\ \forall y : \mathbf{int} \quad x = y) \quad :\quad \mathbf{Boole} \tag{12}$$

It is important to note that (12) merely states that a certain formula is well formed. It does not state that the formula is true. This formula is false.

# 3 Function and Predicate Expressions

It is important to have a way of forming new functions and predicates beyond those that are provided in the foundation. Suppose for example that we want to define the square function $f : \mathbf{int} \to \mathbf{int}$ with the property that for $x : \mathbf{int}$ we have that $f(x) = x * x$. We allow the square function to be written as a lambda expression as follows.

$$\lambda(x : \mathbf{int})\ \ x * x$$

We have the following type inference rule for typing $\lambda$-expressions.

$$\text{A19}\qquad \frac{\Sigma, x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \sigma}{\Sigma \vdash (\lambda(x_1 : \tau_1, \ldots, x_n : \tau_n)\ e) : (\tau_1 \times \cdots \times \tau_n) \to \sigma}$$

Note that the type inference rule for $\lambda$ allows for the formation of both functions, such as the square function, and predicates such as the following even predicate.

$$\vdash (\lambda(x : \mathbf{int})\ \exists y : \mathbf{int}\ x = y + y) : (\mathbf{int} \to \mathbf{Boole})$$

# 4 Free and Bound Variables

The meaning of an expression depends on the meaning of its variables. For example the expression $x = y$ has a well defined truth value when $x$ and $y$ can be assigned the same type and when we assign a particular value to the variable $x$ and the variable $y$. The formula $\forall x : \mathbf{int} \forall y : \mathbf{int}\ x = y$ has a well defined truth value (false) which does not depend on any external assignment of values to $x$ and $y$. We say that the occurrences of $x$ and $y$ in this formula are *bound* by the quantifiers. More formally, any variable occurrence which occurs inside a quantifier binding that variable is called a bound occurrence. So far we have the three quantifiers $\forall$, $\exists$, and $\lambda$. Note that the meaning of the square function $\lambda(x : \mathbf{int})\ x * x)$ does not depend on any particular value of $x$ — the expression denotes a well defined function form integers to integers. An occurrence which is not bound is called *free*.

# 5 Definitions

We now extend the structural axioms A4 and A5 to allow for definitions as well as type declarations. A definition specifies a particular value for a symbol rather than just a type. A definition has the form $x = e$ where $x$ is a symbol and $e$ is a well formed expression. We say that $x$ is defined in $\Sigma$ if $\Sigma$ contains a definition of the form $x = e$. We have the following rule for introducing definitions.

$$\text{A20}\quad \frac{\begin{array}{l}\Sigma \vdash e : \tau \\ x \text{ is a symbol} \\ x \text{ does not occur in } \Sigma \\ x \text{ does not occur in any foundational axiom} \\ \text{every free variable of } e \text{ is defined in } \Sigma\end{array}}{\Sigma\ ;\ x = e \vdash x : \tau}$$

For example we can derive the following sequents.

$$\text{square} = (\lambda(x : \mathbf{int})\ x * x) \quad \vdash \quad \text{square} : \mathbf{int} \to \mathbf{int}$$

$$\text{even} = (\lambda(x : \mathbf{int})\ \exists y : \mathbf{int}\ \ x = y * y) \quad \vdash \quad \text{even} : \mathbf{int} \to \mathbf{Boole}$$

In practice $\Sigma$ is generally taken to contain a very large set of mathematical definitions.

# 6 Semantics

The inference rules A3, A4, and A20 specify the notion of a well formed assumption set $\Sigma$. We have that $\Sigma$ is well formed if there exists some type declaration

$x : \tau$ such that we have $\Sigma \vdash x : \tau$. Any well formed $\Sigma$ contains definitions and type declarations. As such it specifies a set of possible semantic values that can be assigned to the defined and declared variables. Each defined variable has a uniquely defined value — the value of the expression which has been defined to be. Each type-declared variable has a set of possible values given by its type. A *variable interpretation* for $\Sigma$ is an assignment of a value to each variable declared or defined in $\Sigma$ where each defined variable has the specified value and each declared variable is assigned an element of the declared type.

If $\Sigma \vdash \Phi : \mathbf{Boole}$ then we write $\Sigma \models \Phi$ to mean that $\Phi$ is true under any variable interpretation for $\Sigma$. If every free variable of $\Phi$ is defined in $\Sigma$ then these definitions determine the value of $\Phi$. In that case we have either $\Sigma \models \Phi$ or $\Sigma \models \neg\Phi$. Intuitively, if every term is properly defined, and there are no undefined free variables, then we have a well formed mathematical conjecture which is either true or false.

# 7 Definite Descriptions

Suppose that we want to define a square root function over the reals. We can define the square root of $x$ to be *the* unique non-negative value $y$ such that $y * y = x$. It turns out that this form of definition is very convenient. The general definite description has the form $\text{the}(x : \tau\ \Phi)$ where $\Phi$ is a Boolean expression involving $x$. While this is very convenient, we want it to be well formed only when there exists a unique such $x$. We give the following type inference rule for definite descriptions.

$$\text{A21}\quad \frac{\begin{array}{l}\Sigma \vdash P : \tau \to \mathbf{Boole} \\ \Sigma \models \exists x : \tau\ \ P(x) \\ \Sigma \models \forall x : \tau \forall y : \tau\ \ P(x) \wedge P(y) \to x = y\end{array}}{\Sigma \vdash \text{the}(x : \tau\ P(x)) : \tau}$$

Here we are concerned with defining the notion of a well formed mathematical statement. We are not concerned with methods of determining truth. Rather we assume that truth is well defined for well formed statements. The rule A21 assumes that truth is well defined and requires that certain statements are true in order for a certain expression to be well formed. This is consistent with standard mathematical practice.

# 8 Definite Descriptions of Functions and Predicates

Suppose that we want to define the predicate $P : \mathbf{int} \to \mathbf{Boole}$ such that $P(x)$ is true if and only if $x > 0$. Note that $P$ satisfies that following formula.

$$P(1) \wedge \neg P(0) \wedge \forall x : \mathbf{int}\ \ P(x) \Rightarrow P(x+1)$$

Furthermore it is easy to see that the positive predicate is the only predicate satisfying this condition. Hence the following definite description is well formed.

$$\text{the}(P : \mathbf{int} \rightarrow \mathbf{Boole}\ \ P(1) \wedge \neg P(0) \wedge \forall x : \mathbf{int}\ \ P(x) \Rightarrow P(x+1))$$

Since this definite description is well formed, we can use it as the definition of the predicate positive

$$\text{positive} = \text{the}(P : \mathbf{int} \rightarrow \mathbf{Boole}\ \ P(1) \wedge \neg P(0) \wedge \forall x : \mathbf{int}\ \ P(x) \Rightarrow P(x+1))$$

We can then define the binary relation $\geq$ as follows.

$$\geq = (\lambda\ (x : \mathbf{int},\ y : \mathbf{int})\ x = y \vee \text{positive}(x + (-1 * y)))$$

Recursive definitions can in general be handled using definite descriptions. For example consider the function exp on two integers defined by $\exp(x, n) = x^n$ if $n > 0$ and 1 otherwise. We can define exp as follows.

exp=
the($f$: **int** $\times$ **int** $\rightarrow$ **int**
$\forall x$ : **int** $\forall m$ : **int**
$m \leq 0 \Rightarrow f(x, m) = 1$
$\wedge\ \ m > 0 \Rightarrow f(x, m) = x * f(x, m-1)$)