Newman's Typability Algorithm

David MacQueen

WG2.8 Keffalonia, 2015

Max Newman (1897 - 1984)

The Papers

1910: Alfred North Whitehead and Bertrand Russell, Principia Mathematica

1937: Willard Van Orman Quine, New Foundations for Mathematical Logic

1940: Alonzo Church, A formulation of the simple theory of types

1943: M. H. A. Newman, **Stratified Systems of Logic**

2006: J. Roger Hindley, M. H. Newman's Typability Algorithm for Lambda-Calculus

2011: J. H. Geuvers and R Krebbers, The correctness of Newman's typability algorithm and some of its extensions

Quine's New Foundations

A simplified reworking of Whitehead and Russell's explicitly typed set theory.

Key Ideas:

1. The axiom of comprehension is restricted to stratified formulas.
2. Stratification of a formula is an implicit, inferred property (essentially, the existence of a consistent *typing* of variables occurring in subformulas of the form $x \in y$).

Thus Quine is to Whitehead and Russell as Curry is to Church.

Stratified Formulas in NF

Quine, 1937:

> The rule is imposed, finally, that $(\alpha \in \beta)$ is to be a formula only if the values of β are of next higher type than those of α; otherwise $(\alpha \in \beta)$ is reckoned as neither true nor false, but meaningless.
>
> In all contexts the types appropriate to the several variables are actually left unspecified; the context remains systematically ambiguous, in the sense that the types of its variables may be construed in any fashion conformaable to the requirement that "$\in$" connect variables only of consecutively ascending types. An expression which would be a formula under our original scheme will hence be rejected as meaningless by the theory of types only if there is now way whatever of so assigning types to the variables as to conform to this requirement on "$\in$".
>
> Formulas passing this test will be called *stratified.*

Syntactic test for stratification in NF

Quine gives a syntactic way of characterizing stratified formulas as follows:

> An $\in$-chain of ϕ is an expression $\alpha_1 \in \alpha_2 \in \alpha_3 \cdots \in \alpha_n$ $(n > 1)$ such that each segment $\alpha_i \in \alpha_{i+1}$ occurs in ϕ. Now ϕ is *stratified* if it has no $\in$-chains with like initial and like terminal variables but unlike lengths.

This was shown to be incorrect by Bernays, but Newman gives a correct replacement, namely that there are no $\in$-chains $\alpha_1 \in \alpha_2 \in \alpha_3 \cdots \in \alpha_1$ $(n > 1)$ where the first and last variables are the same.

Newman also gives an algorithm for deciding whether a formula of NF is stratified as an example of a very general and abstract scheme that also can be applied to the untyped lambda calculus. And he shows that stratification is equivalent to typability, for appropriate notions of typability.

Newman, 1943

The paper begins with the statement:

> The suffixes used in logic to indicate differences of type may be regarded either as belonging to the formalism itself, or as being part of the machinery for deciding which rows of symbols (without suffixes) are to be admitted as significant.

Again, Church vs Curry typing!

Newman's Algorithm for λ-calculus

Start with a pure lambda calculus expression, possibly containing free variables. Assume all bound variables are different from one another and from the free variables.

Name each compound subexpression by breaking the expression into a set of equations. (Variables name themselves.)

Example

$$u(ux) \quad \Longrightarrow \quad M = uP \qquad P = (ux)$$

Define two binary relations $>_d$ and $>_r$ on expression names:

$$X = PM \quad \Longrightarrow \quad \begin{array}{c} P \ >_d \ M \\ P \ >_r \ X \end{array} \qquad t(P) = t(M) \to t(X)$$

$$X = \lambda x.M \quad \Longrightarrow \quad \begin{array}{c} X \ >_d \ x \\ X \ >_r \ M \end{array} \qquad t(X) = t(x) \to t(M)$$

Here we are viewing expression metavariables as standing for both the subexpression, and its type.

Define $\sim$ as the equivalence closure of the relation $\sim_0$ defined by:

$$\begin{array}{c} P >_d M \\ P >_d N \end{array} \quad \Longrightarrow \quad M \sim_0 N \qquad \text{(domain of common function)}$$

$$\begin{array}{c} P >_r M \\ P >_r N \end{array} \quad \Longrightarrow \quad M \sim_0 N \qquad \text{(range of common function)}$$

$$\begin{array}{c} M >_d X, \quad N >_d X \\ M >_r Y, \quad N >_r Y \end{array} \quad \Longrightarrow \quad M \sim_0 N \qquad \text{(same domain and range)}$$

The relation $\sim$ corresponds roughly to unification.

Let $[M]$ be the $\sim$ equivalence class of M.

Defn: $[M] >_d [N]$ if $M >_d N$

Let $> \quad = \quad >_d \cup >_r$

An expression is *stratified* if there are no $>$ cycles.

In other words, if $>^*$ is a strict partial order. This corresponds to the occurrence check in unification.

Theorem 4. An expression is typable if and only if it is stratified.

Example 1: $M \; = \; u(ux)$

$$M = uP, \quad P = ux \tag{1}$$

$$u >_r M, \quad u >_d P, \quad u >_r P, \quad u >_d x \tag{2}$$

$$u >_r M, \;\; u >_r P \quad \Longrightarrow \quad M \sim P \tag{3}$$

Replace P by M in (2) and eliminate duplicates:

$$u >_r M, \quad u >_d M, \quad u >_d x \tag{4}$$

$$u >_d M, \;\; u >_d x \quad \Longrightarrow \quad x \sim M \tag{5}$$

Replace M by x in (4) and eliminate duplicates:

$$u >_r x, \quad u >_d x \tag{6}$$

No $>$-cycles, hence $u(ux)$ is stratified, hence typable.

Example 2: $M \;=\; \lambda x.ux$

$$M = \lambda x.P, \quad P = ux \tag{1}$$

$$M >_r P, \quad u >_r P, \quad M >_d x, \quad u >_d x \tag{2}$$

(2) implies $u \sim M$ (common ranges and domains), so replace M with u.

$$u >_r P, \quad u >_d x \tag{3}$$

No $>$-cycles, therefore $\lambda x.ux$ is stratified, hence typable.

Example 3: $M \;=\; xx$

$$M = xx \tag{1}$$

$$x >_r M, \quad x >_d x \tag{2}$$

$x >_d x$ is a $>$-cycle, hence xx is not stratified, hence not typable.

Type Inference

The proof of sufficiency (*i.e.*, stratified implies typable) for Theorem 4 implicitly determines a procedure for constructing a type for a stratified term. We can start by assigning distinct type variables to the $>$-minimal expressions, and then work back through the $>_d$ and $>_r$ relations to assign types to the rest of the subterms, with all members of a $\sim$-equivalence class assigned the same type.