Newman's Typability Algorithm

Herman Geuvers[1]

Radboud University Nijmegen
and
Eindhoven University of Technology
The Netherlands

Computing 2011
Symposium on 75 Years of Turing Machine and Lambda-Calculus
Karlsruhe, October 2011

[1] Joint work with Robbert Krebbers, RU Nijmegen

First half of the 20th century

Questions:

- What can be computed? What can be decided?
- What is a good foundation for logic and mathematics?
  ... higher-order logic, set theory, ... without inconsistencies

First half of the 20th century

Questions:

- What can be computed? What can be decided?
- What is a good foundation for logic and mathematics?
  ... higher-order logic, set theory, ... without inconsistencies

Begriffschrift (Frege), Principia Mathematica (Whitehead, Russell), Axiomatic Set theory (Zermelo, Fraenkel), Theory of simple types (Church), New Foundations (Quine), ...

First half of the 20th century

Questions:

- What can be computed? What can be decided?
- What is a good foundation for logic and mathematics?
  ... higher-order logic, set theory, ... without inconsistencies

Begriffschrift (Frege), Principia Mathematica (Whitehead, Russell), Axiomatic Set theory (Zermelo, Fraenkel), Theory of simple types (Church), New Foundations (Quine), ...

- Which "objects" exist? ($\{x \mid x \notin x\}$ does not ...)
- Which "terms" are well-formed? ($F(F)$, for F a property?)

Distinction between syntax and semantics?

Newman's article 1943

PROCEEDINGS OF THE
Cambridge Philosophical Society

VOL. 39 — *June* 1943 — PART 2

STRATIFIED SYSTEMS OF LOGIC

BY M. H. A. NEWMAN

Received 2 September 1942

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. The two different attitudes do not necessarily lead to different formalisms, but when types are regarded as only one way of regulating the calculus it is natural to consider other possible ways, in particular the direct characterization of the significant formulae. Direct criteria for stratification were given by Quine, in his 'New Foundations for Mathematical Logic'(7). In the corresponding typed form of this theory ordinary integers are adequate as type-suffixes, and the direct description is correspondingly simple, but in other theories, including that recently proposed by Church(4), a partially ordered set of types must be used. In the present paper criteria, equivalent to the existence of a correct typing, are given for a general class of formalisms, which includes Church's system, several systems proposed by Quine, and (with some slight modifications, given in the last paragraph) *Principia Mathematica*. (The discussion has been given this

M.H.A. Newman, 1897 – 1984

- English topologist with side-interest in logic and foundations of mathematics
- Newman's Lemma
  "On theories with a combinatorial definition of equivalence". Annals of Mathematics, 1942.

  > *If the binary relation R is weakly confluent and terminating, then R is confluent.*

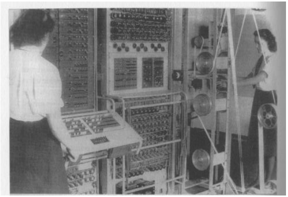
M.H.A. Newman, 1897 – 1984

Wrote "Stratified Systems of Logic" in 1943

- Abstract algorithm to decide typability
- Quine's "New Foundation" was his starting point
- Also works on a variant of simple type theory $\lambda\!\rightarrow$ of Church [1940].
- Returns true or false instead of a principal type
- At first sight very different from the standard algorithm

M.H.A. Newman, Relation with Turing and computability

- Taught and inspired Turing at Cambridge (1930s)
- Worked in Bletchley Park, on Colossus (1942 onwards), the first “real” computer
- Appointed Turing at math. dept. in Manchester
- His original name was “Neumann” ...



Roger Hindley

- J.R. Hindley: M.H. Newman's typability algorithm for lambda-calculus, *J. Logic and Computation* 18(2): 229-238 (2008) 17.
  (Talk at Jan Willem Klop's 60th Birthday, 2005)
- Question: How does Newman's algorithm compare to the standard typability algorithm?

Roger Hindley

- J.R. Hindley: M.H. Newman's typability algorithm for lambda-calculus, *J. Logic and Computation* 18(2): 229-238 (2008) 17.
  (Talk at Jan Willem Klop's 60th Birthday, 2005)
- Question: How does Newman's algorithm compare to the standard typability algorithm?
- The correctness of Newman's typability algorithm and some of its extensions, H. G., Robbert Krebbers *TCS* 412, 2011.

Newman was ahead of his time

In 1944 Church reviewed Newman's paper in JSL, concluding:

> *The reader's first impression of Newman's paper may be that the machinery introduced is heavy in comparison with the results obtained. The value of the paper is in fact difficult to estimate at present, as this will depend on the extent to which results obtained in the future by Newman's methods justify the weight of machinery.*

Simple Type Theory à la Curry

Assign types to untyped λ-terms.

$$\begin{array}{rcl} \Lambda & ::= & V \mid (\Lambda\,\Lambda) \mid (\lambda V.\Lambda) \\ T & ::= & \mathsf{TypeVar} \mid T \to T \end{array}$$

Contexts: $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ ($x_i \in V$, $\sigma_i \in T$)

$$\text{(var)} \quad \Gamma \vdash x : \sigma \qquad\qquad \text{if } x : \sigma \in \Gamma$$

$$\text{(app)} \quad \frac{\Gamma \vdash M : \sigma \to \tau \qquad\qquad \Gamma \vdash P : \sigma}{\Gamma \vdash M\,P : \tau}$$

$$\text{(abs)} \quad \frac{\Gamma, x : \sigma \vdash N : \tau}{\Gamma \vdash \lambda x.N : \sigma \to \tau}$$

Simple Type Theory à la Newman

$$\Lambda \ ::= \ V \mid (\Lambda\,\Lambda) \mid (\lambda V.\Lambda)$$

Only terms that satisfy the Barendregt convention: So, in a term:

- all bound variables are different from the free ones
- all bound variables are different.

Simple Type Theory à la Newman

$$\Lambda \quad ::= \quad V \mid (\Lambda\,\Lambda) \mid (\lambda V.\Lambda)$$

Only terms that satisfy the Barendregt convention: So, in a term:

- all bound variables are different from the free ones
- all bound variables are different.

Example

Not $x\,(\lambda x.x)$
Not $(\lambda x.x)\,(\lambda x.x)$

Newman's algorithm: Schemes

Newman's algorithm is a system for rewriting the scheme of a term. A scheme over a domain A and set of operation symbols Φ consists of

A finite list of equations of the form.

$$X \doteq \varphi X_1 X_2 \dots X_{\mathsf{ar}(\varphi)} \qquad \textit{where } X, X_i \in A \textit{ and } \varphi \in \Phi$$

The (finitely many) operation symbols in Φ have a fixed arity $\mathsf{ar} : \Phi \to \mathbb{N}$.

Newman's algorithm: Scheme of a λ-term

The domain is

$$\mathsf{Name} ::= \mathsf{TermName} \mid \mathsf{Var}$$

Equations are of the form

$$\mathsf{Name} \doteq \mathsf{app}\ \mathsf{Name}\ \mathsf{Name} \qquad \mathsf{Name} \doteq \lambda\ \mathsf{Name}\ \mathsf{Name}$$

As notation, we of course just use

$$\mathsf{Name} \doteq \mathsf{Name}\ \mathsf{Name} \qquad \mathsf{Name} \doteq \lambda \mathsf{Name}.\mathsf{Name}$$

Newman's algorithm: Scheme of a λ-term

The domain is

$$\text{Name} ::= \text{TermName} \mid \text{Var}$$

Equations are of the form

$$\text{Name} \doteq \text{app Name Name} \qquad \text{Name} \doteq \lambda \text{ Name Name}$$

As notation, we of course just use

$$\text{Name} \doteq \text{Name Name} \qquad \text{Name} \doteq \lambda\text{Name}.\text{Name}$$

$\mathcal{S}(M)$ generates a list of equation of this form from M

Example

The scheme $\mathcal{S}(M)$ of $M \equiv \lambda fx.f(fx)$ is

$$U \doteq \lambda f.V \qquad V \doteq \lambda x.W \qquad W \doteq fZ \qquad Z \doteq fx$$

Newman's algorithm: Reduction of a scheme

$$\begin{array}{ccccc} M & \rightarrow & S_1 = \mathcal{S}(M) & \rightarrow & \text{binary relations } \eta \text{ and } \gamma \\ & & \downarrow & \text{if } X \; \eta \; Y & \\ & & S_2 = S_1[X := Y] & & \end{array}$$

Newman's algorithm: Reduction of a scheme

$$
\begin{array}{ccccc}
M & \rightarrow & S_1 = \mathcal{S}(M) & \rightarrow & \text{binary relations } \eta \text{ and } \gamma \\
 & & \downarrow & \text{if } X \; \eta \; Y & \\
 & & S_2 = S_1[X := Y] & \rightarrow & \text{binary relations } \eta \text{ and } \gamma \\
 & & \vdots & & \\
 & & \downarrow & & \\
 & & S_f & &
\end{array}
$$

S_f is the η-normal form (no more η-reduction exists)

(NB: This is something completely different from the well-known η-reduction in λ-calculus.)

Newman's algorithm: Stratification

Definition

A scheme S is stratified iff no cycles in the γ-relation exist.

Newman's result:

Let $M \in \Lambda$ and $\mathcal{S}(M) \rightarrow_{\eta} S_f$ (in normal form).

Then S_f is stratified iff M is typable.

Newman's algorithm: Properties

- Reduction is strongly normalizing
- Reduction is locally confluent up to renaming of letters
- Thus the result is unique up to renaming of letters
- Whether S_f is stratified is independent of the order of reduction

From Lambda Trees to "Newman Graphs"

A Modern presentation of Newman's algorithm

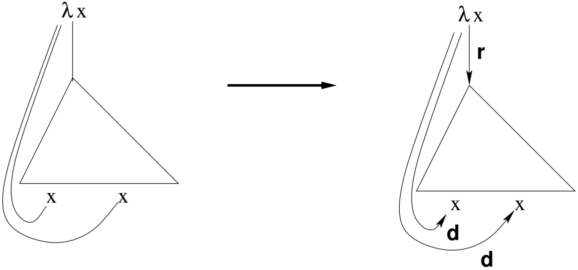
$U \xrightarrow{d} V$: the type of V is the domain of the type of U.

$U \xrightarrow{r} V$: the type of V is the range of the type of U.

From Lambda Trees to "Newman Graphs"

A Modern presentation of Newman's algorithm

$U \xrightarrow{d} V$: the type of V is the domain of the type of U.
$U \xrightarrow{r} V$: the type of V is the range of the type of U.

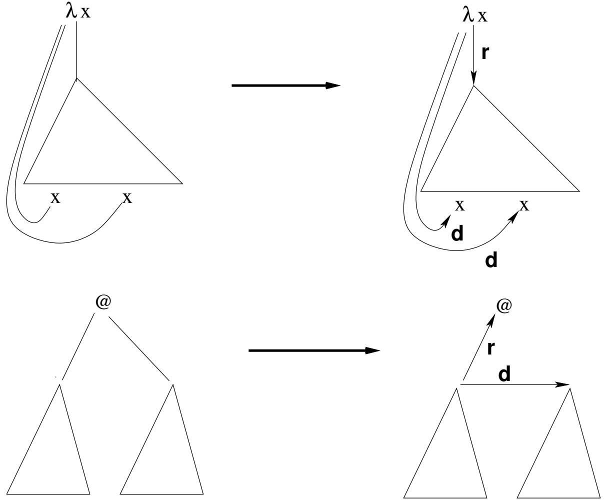


From Lambda Trees to "Newman Graphs"

A Modern presentation of Newman's algorithm

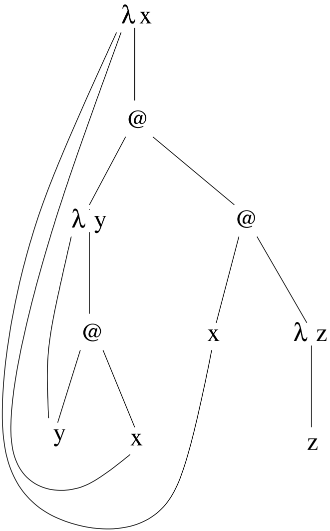
$U \xrightarrow{d} V$: the type of V is the domain of the type of U.

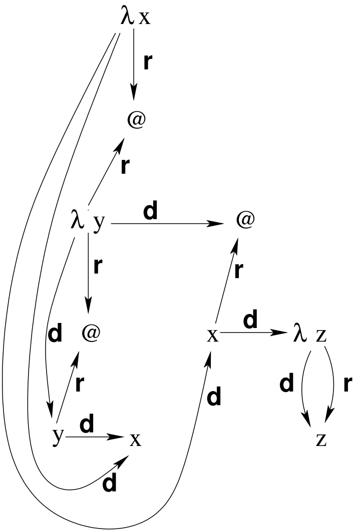
$U \xrightarrow{r} V$: the type of V is the range of the type of U.



Example

$$\lambda x.\,(\lambda y.\,y\,x)\,(x\,(\lambda z.\,z))$$





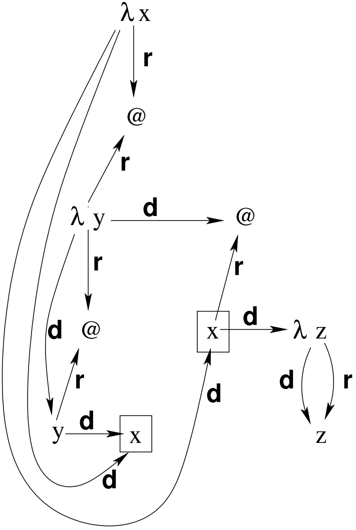
Equivalence Relation on Nodes $\simeq$

$$\begin{tikzcd} & X \\ U \arrow[ur, "d"] \arrow[dr, "d"'] & \\ & Y \end{tikzcd} \quad \Longrightarrow \quad X \simeq Y \quad (A)$$

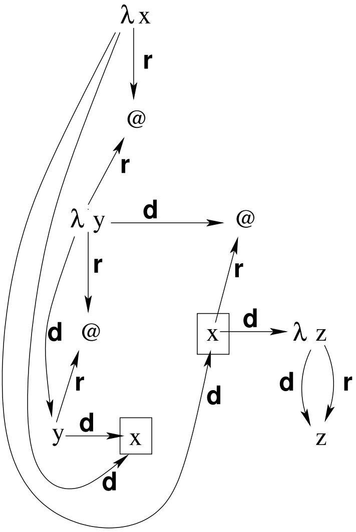
$$\begin{tikzcd} & X \\ U \arrow[ur, "r"] \arrow[dr, "r"'] & \\ & Y \end{tikzcd} \quad \Longrightarrow \quad X \simeq Y \quad (A)$$

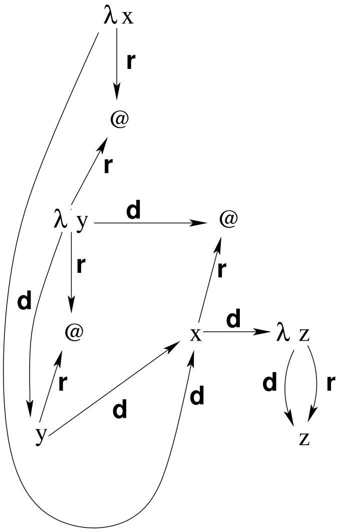
$$\begin{tikzcd} X \arrow[dr, "r"] & \\ & U \\ Y \arrow[ur, "r"'] & \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} X \arrow[dr, "d"] & \\ & V \\ Y \arrow[ur, "d"'] & \end{tikzcd} \quad \Longrightarrow \quad X \simeq Y \quad (B)$$

Example: Joining Equivalent Nodes

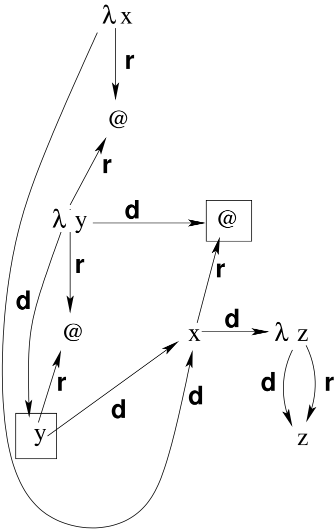


Example: Joining Equivalent Nodes

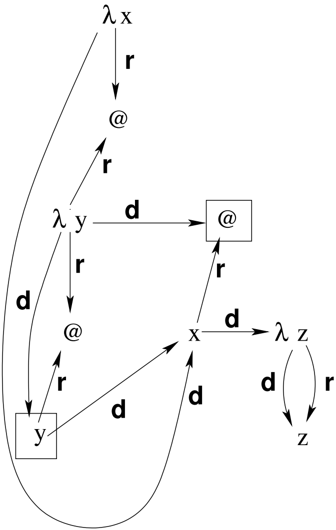


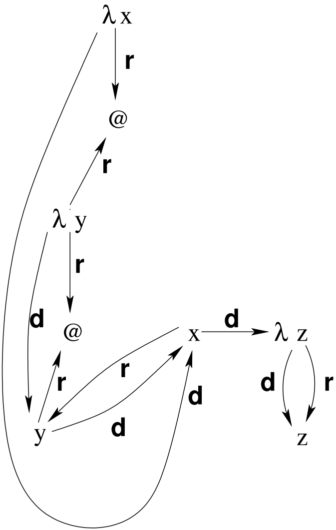


Example: Joining Equivalent Nodes

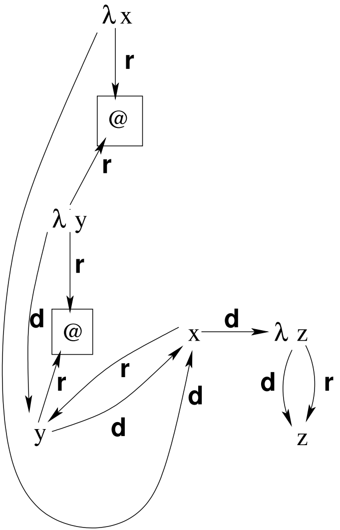


Example: Joining Equivalent Nodes

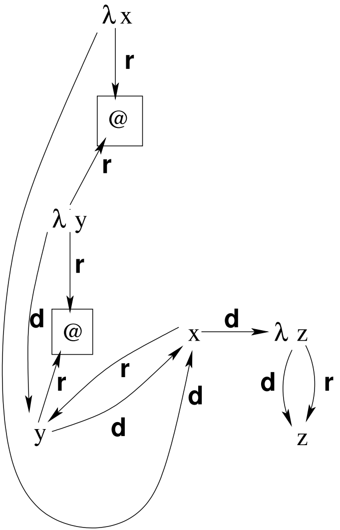




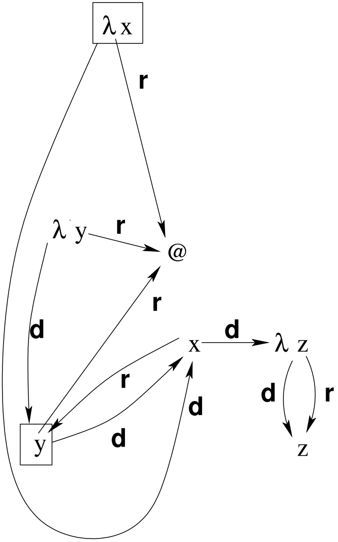
Example: Joining Equivalent Nodes



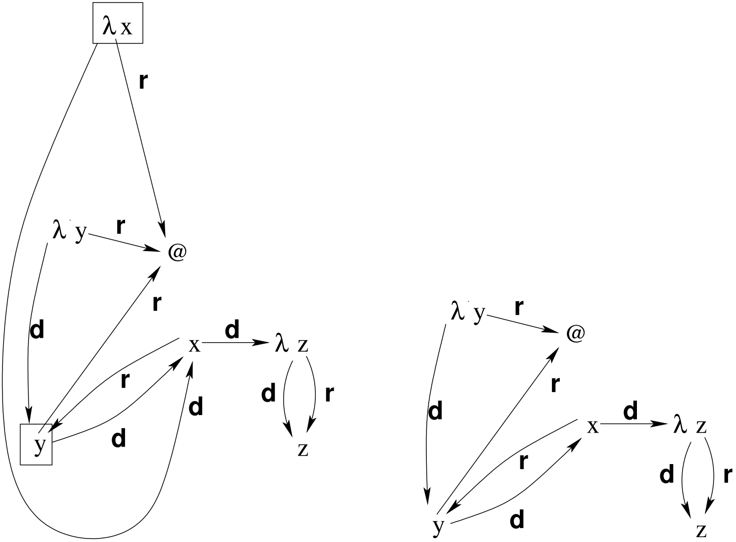
Example: Joining Equivalent Nodes



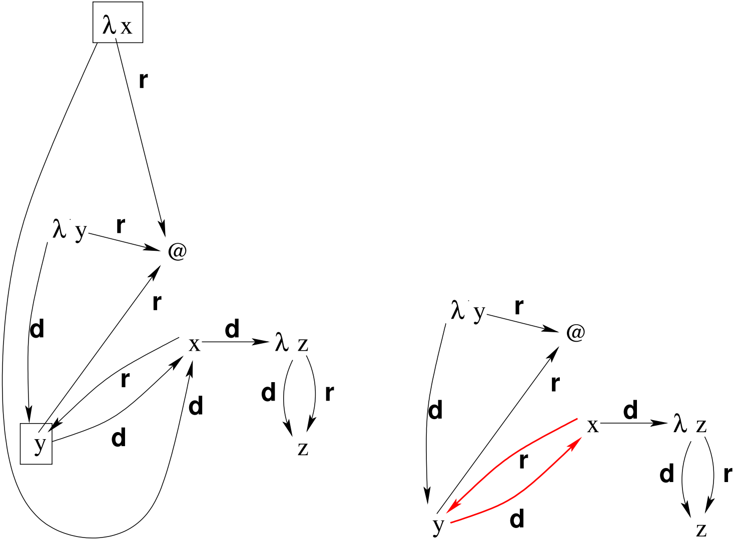
Example: Joining Equivalent Nodes



Example: Joining Equivalent Nodes



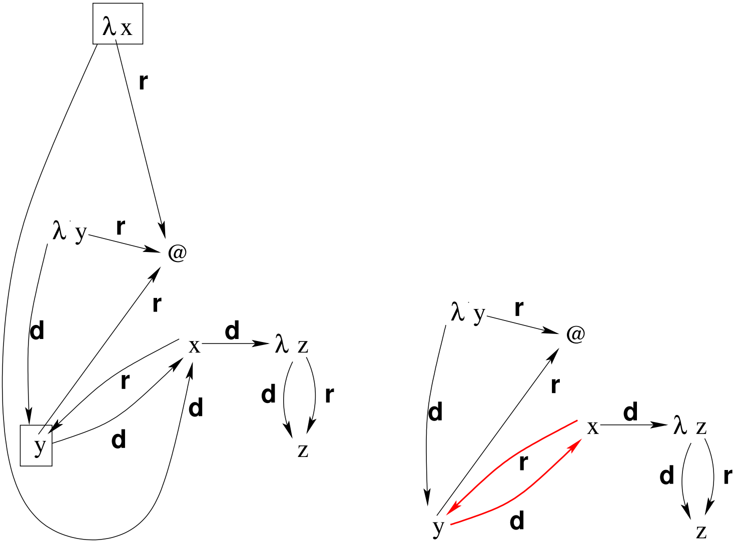
Example: Joining Equivalent Nodes



The graph is in normal form.

It contains a cycle, so the term is not typable.

Example: Joining Equivalent Nodes



The graph is in normal form.

It contains a cycle, so the term is not typable (Theorem).

Newman's algorithm: original form

Definition

Given a scheme S of a λ-term, define the relations γ_d and γ_r over Name as follows.

$$
\begin{aligned}
Z \doteq MN \quad &\Longrightarrow \quad M \;\gamma_d\; N \wedge M \;\gamma_r\; Z \\
Z \doteq \lambda x.P \quad &\Longrightarrow \quad Z \;\gamma_d\; x \wedge Z \;\gamma_r\; P
\end{aligned}
$$

Newman's algorithm: original form

Definition

Given a scheme S of a λ-term, define the relations γ_d and γ_r over Name as follows.

$$
\begin{aligned}
Z \doteqdot MN \quad &\Longrightarrow \quad M\ \gamma_d\ N \wedge M\ \gamma_r\ Z \\
Z \doteqdot \lambda x.P \quad &\Longrightarrow \quad Z\ \gamma_d\ x \wedge Z\ \gamma_r\ P
\end{aligned}
$$

Definition

Given a scheme S, define the binary relation η as follows.
$X\ \eta\ Y$ iff one of the following conditions hold:

1. $\exists_{U \in A} \exists_{\gamma_i} [U\ \gamma_i\ X \wedge U\ \gamma_i\ Y]$
2. $\forall_{\gamma_i} \exists_{U \in A} [X\ \gamma_i\ U \wedge Y\ \gamma_i\ U]$

$X\ \gamma\ Y$ iff $\exists_{\gamma_i} [X\ \gamma_i\ Y]$

Newman's algorithm: η-reduction

Definition

An η-reduction in a scheme S replaces X in all equations by Y if $X \neq Y$ and $X \;\eta\; Y \in S$.
Notation: $S \overset{X:=Y}{\rightarrow_\eta} S'$, multiple steps are denoted by $S \overset{\nu}{\twoheadrightarrow_\eta} S'$ where ν is a substitution.

Lemma

η-reduction is strongly normalising.

Definition

A scheme S is stratified iff no cycles in the γ-relations of S exist.

Theorem

The η-normal form of $\mathcal{S}(()M)$ is stratified iff M is typable.

Newman's algorithm: η-reduction

Example

Take the scheme $S = \mathcal{S}(M)$ of the λ-term $M \equiv f(fx)$.

$$W \doteqdot fZ \qquad Z \doteqdot fx$$

$$\boxed{f\ \gamma_d\ Z} \qquad f\ \gamma_r\ W \qquad \boxed{f\ \gamma_d\ x} \qquad f\ \gamma_r\ Z$$

Newman's algorithm: η-reduction

Example

Take the scheme $S = \mathcal{S}(M)$ of the λ-term $M \equiv f(fx)$.

$$W \doteq fZ \qquad Z \doteq fx$$

$$\boxed{f\ \gamma_d\ Z} \qquad f\ \gamma_r\ W \qquad \boxed{f\ \gamma_d\ x} \qquad f\ \gamma_r\ Z$$

After one step of η-reduction, $S \stackrel{Z:=x}{\to}_\eta S'$, the scheme S' is obtained.

$$W \doteq fx \qquad x \doteq fx$$

$$f\ \gamma_d\ x \qquad \boxed{f\ \gamma_r\ W} \qquad \boxed{f\ \gamma_r\ x}$$

Newman's algorithm: η-reduction

Example

Take the scheme $S = \mathcal{S}(M)$ of the λ-term $M \equiv f(fx)$.

$$W \doteqdot fZ \qquad Z \doteqdot fx$$

$$\boxed{f\ \gamma_d\ Z} \qquad f\ \gamma_r\ W \qquad \boxed{f\ \gamma_d\ x} \qquad f\ \gamma_r\ Z$$

After one step of η-reduction, $S \overset{Z:=x}{\to_\eta} S'$, the scheme S' is obtained.

$$W \doteqdot fx \qquad x \doteqdot fx$$

$$f\ \gamma_d\ x \qquad \boxed{f\ \gamma_r\ W} \qquad \boxed{f\ \gamma_r\ x}$$

Finally an η-irreducible scheme S_f is obtained by $S' \overset{W:=x}{\to_\eta} S_f$.

$$x \doteqdot fx \qquad x \doteqdot fx$$

$$f\ \gamma_d\ x \qquad f\ \gamma_r\ x$$

Relation to the standard typing algorithm: Wand

Wand's algorithm produces a scheme of type equations.
These are solved using unification.
SG: set of goals: triples (Γ, M, σ)
EQ: set of equations: $\sigma = \tau$

Relation to the standard typing algorithm: Wand

Wand's algorithm produces a scheme of type equations.
These are solved using unification.
SG: set of goals: triples (Γ, M, σ)
EQ: set of equations: $\sigma = \tau$
Action table:

g	$SG(g)$	$EQ(g)$
(Γ, x, τ)	$\emptyset$	$\tau = \Gamma(x)$
$(\Gamma, \lambda x.M, \tau)$	$(\Gamma; x : \alpha_1, M, \alpha_2)$	$\tau = \alpha_1 \to \alpha_2$
$(\Gamma, M\,P, \tau)$	$(\Gamma, M, \alpha \to \tau), (\Gamma, P, \alpha)$	$\emptyset$

Relation to the standard typing algorithm: Wand

Wand's algorithm produces a scheme of type equations.
These are solved using unification.
SG: set of goals: triples (Γ, M, σ)
EQ: set of equations: $\sigma = \tau$
Action table:

g	$SG(g)$	$EQ(g)$
(Γ, x, τ)	$\emptyset$	$\tau = \Gamma(x)$
$(\Gamma, \lambda x.M, \tau)$	$(\Gamma; x : \alpha_1, M, \alpha_2)$	$\tau = \alpha_1 \to \alpha_2$
$(\Gamma, M\, P, \tau)$	$(\Gamma, M, \alpha \to \tau), (\Gamma, P, \alpha)$	$\emptyset$

"Newman's idea": Adapt Wand's algorithm to generate a scheme of equations of the following (simpler) form

$$\mathsf{TVar} \doteq \mathsf{TVar} \to \mathsf{TVar}$$

Relation to the standard algorithm: Wand

Wand's original algorithm:

Action table:

g	$SG(g)$	$EQ(g)$
(Γ, x, τ)	$\emptyset$	$\tau = \Gamma(x)$
$(\Gamma, \lambda x.M, \tau)$	$(\Gamma; x : \alpha_1, M, \alpha_2)$	$\tau = \alpha_1 \to \alpha_2$
$(\Gamma, M\, P, \tau)$	$(\Gamma, M, \alpha \to \tau), (\Gamma, P, \alpha)$	$\emptyset$

Adapted Wand's algorithm

Action table:

g	$SG(g)$	$EQ(g)$
(Γ, x, τ)	$\emptyset$	$\tau = \Gamma(x)$
$(\Gamma, \lambda x.M, \tau)$	$(\Gamma; x : \alpha_1, M, \alpha_2)$	$\tau = \alpha_1 \to \alpha_2$
$(\Gamma, M\, P, \tau)$	$(\Gamma, M, \alpha_1), (\Gamma, P, \alpha_2)$	$\alpha_1 = \alpha_2 \to \tau$

After substituting equations of the form $\tau_1 = \tau_2$ we obtain a scheme of equations of the form

$$\mathsf{TVar} \doteq \mathsf{TVar} \to \mathsf{TVar}$$

Relation to the standard algorithm

- Scheme of type equations (à la Wand)

  $\cong$

  Scheme of λ-term (à la Newman)
- Computation of most general unifier $\cong$ reduction of schemes

Relation to the standard algorithm

- Scheme of type equations (à la Wand)

  $$\cong$$

  Scheme of λ-term (à la Newman)
- Computation of most general unifier $\cong$ reduction of schemes

Corollary

- Newman's algorithm can be extended to compute a principal type / principal pair
- Newman's algorithm is correct

Further remarks and Conclusion

- Newman's method can be extended to incorporate contexts and other type constructions, like sum types, product types and weak polymorphism.
- Basically, Newman gives an efficient unification algorithm:

  All equations are of the form

$$\begin{array}{rcl} X & \doteq & Y \\ X & \doteq & f(Y_1, \ldots, Y_n) \end{array}$$

  where X and $Y_1, \ldots, Y_n$ are variables.