Newman's Typability Algorithm

Herman Geuvers (joint work with Robbert Krebbers)
Radboud University Nijmegen
Technical University Eindhoven
The Netherlands

November 30, 2009

Newman's article

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 general form rather with a view to clarity than to comprehensiveness.)

About M. H. Newman

- English topologist with side-interest in logic
- Newman's Lemma
  "On theories with a combinatorial definition of equivalence".
  Annals of Mathematics, 1942.
- 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 $\lambda\!\rightarrow$ à la Curry
  - Returns *true* or *false* instead of a *principal type*
  - At first sight very different from the standard algorithm

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?

Simple Type Theory à la Curry

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

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

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

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.

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.

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

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

Newman's algorithm: Scheme of a λ-term

The domain is

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

Equations are of the form

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

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

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

$\mathcal{S}(M)$ 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

$$M \quad \rightarrow \qquad S_1 = \mathcal{S}(M) \qquad \rightarrow \quad \text{binary relations } \eta \text{ and } \gamma$$

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 claim

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

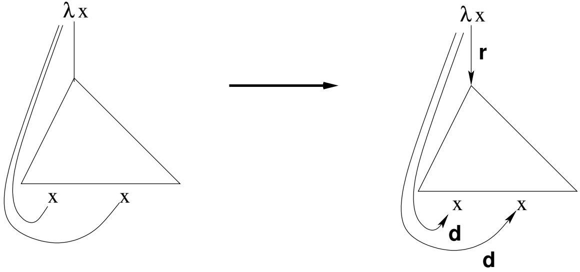
- Reduction is strongly normalizing
- Reduction is locally confluent up to renaming of letters
- Thus the result is unique up to renaming of letters
- Whether S_f is stratified is independent of the order of reduction

From Lambda Trees to “Newman Graphs”

A Modern presentation of Newman’s algorithm

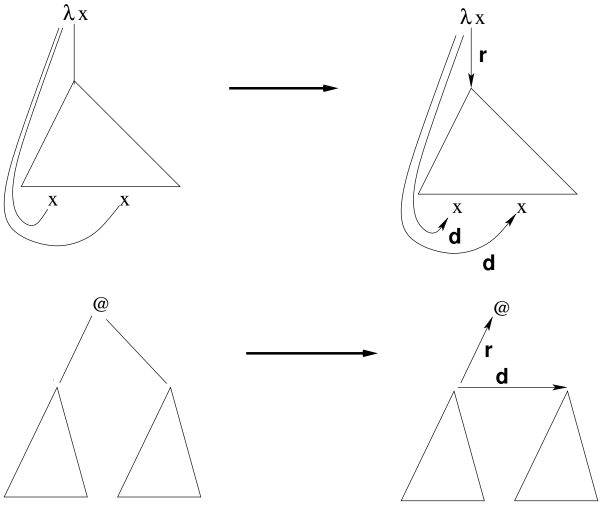
From Lambda Trees to “Newman Graphs”

A Modern presentation of Newman’s algorithm



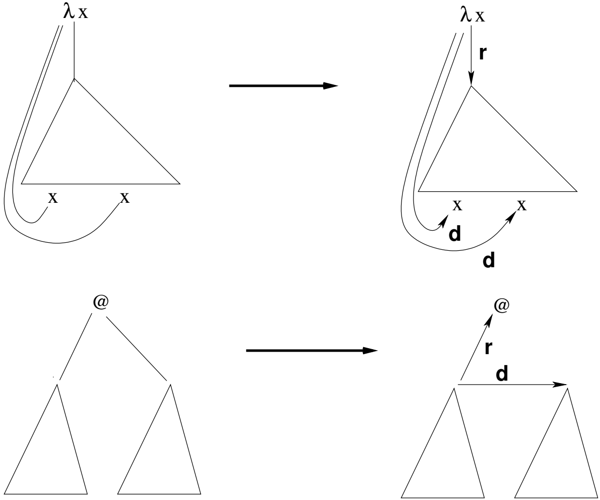
From Lambda Trees to "Newman Graphs"

A Modern presentation of Newman's algorithm



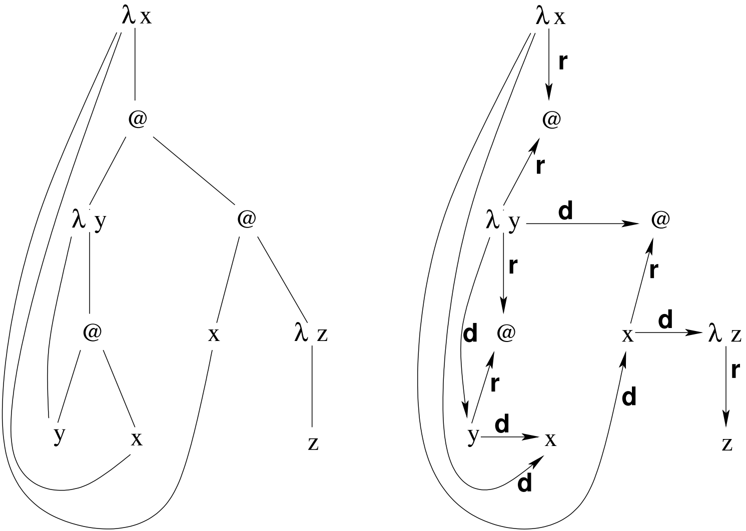
From Lambda Trees to "Newman Graphs"

$U \xrightarrow{d} V$: 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 $\simeq$

$$\begin{tikzcd}[row sep=small] & X \\ U \arrow[ur, "d"] \arrow[dr, "d"'] & \\ & Y \end{tikzcd} \qquad \Longrightarrow \qquad X \simeq Y$$

$$\begin{tikzcd}[row sep=small] & X \\ U \arrow[ur, "r"] \arrow[dr, "r"'] & \\ & Y \end{tikzcd} \qquad \Longrightarrow \qquad X \simeq Y$$

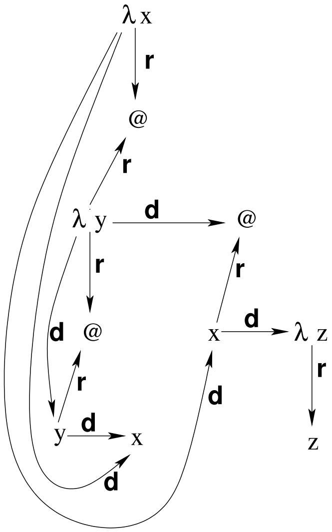
Equivalence Relation on Nodes $\simeq$

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

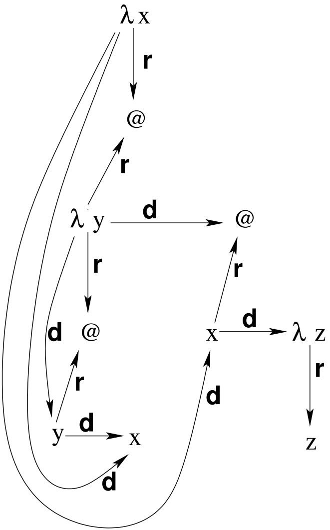
$$
\begin{tikzcd}
 & X \\
U \arrow[ur, "r"] \arrow[dr, "r"'] & \\
 & Y
\end{tikzcd}
\qquad \Longrightarrow \qquad X \simeq Y
$$

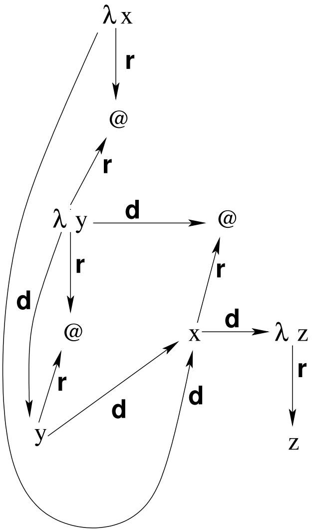
$$
\begin{tikzcd}
X \arrow[dr, "r"] & \\
 & U \\
Y \arrow[ur, "r"'] &
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
X \arrow[dr, "d"] & \\
 & V \\
Y \arrow[ur, "d"'] &
\end{tikzcd}
\qquad \Longrightarrow \qquad X \simeq Y
$$

Joining Equivalent Nodes

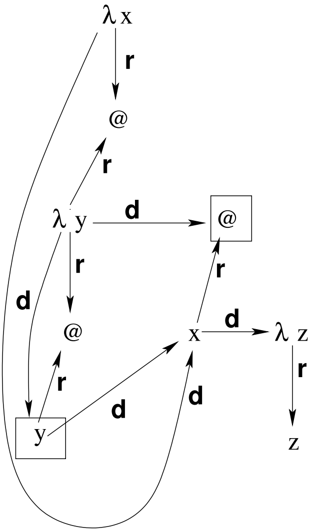


Joining Equivalent Nodes

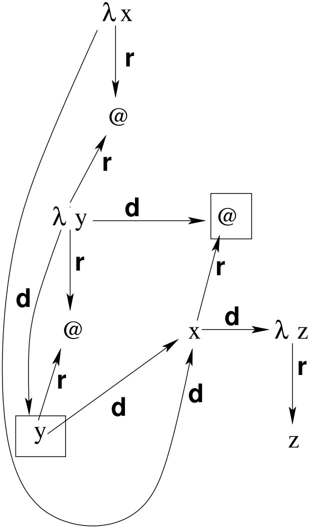


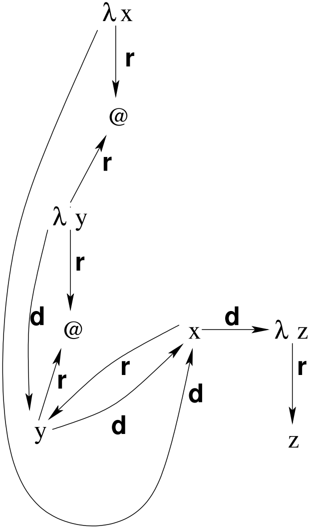


Joining Equivalent Nodes

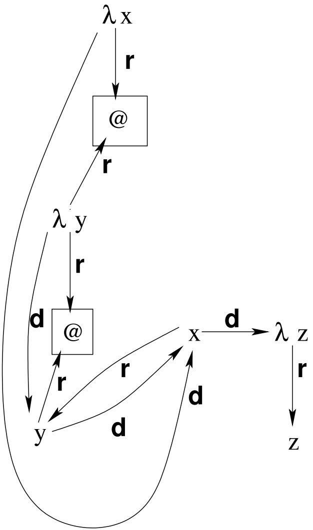


Joining Equivalent Nodes

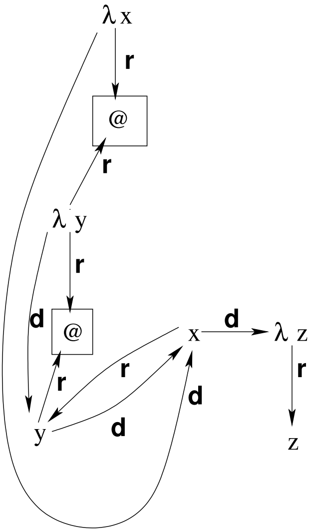


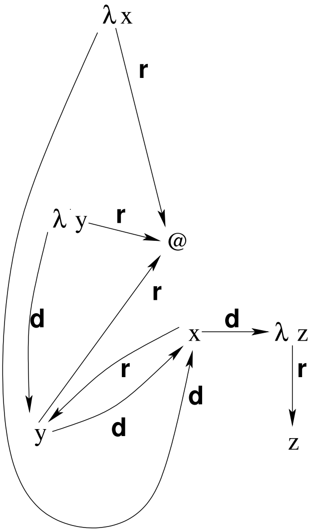


Joining Equivalent Nodes

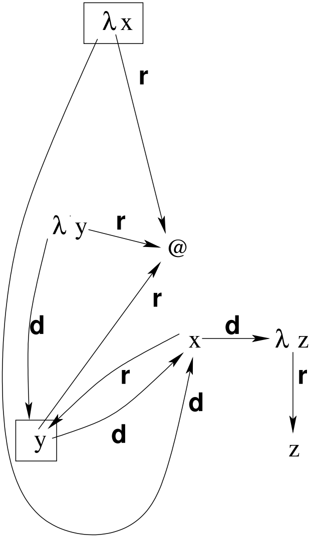


Joining Equivalent Nodes

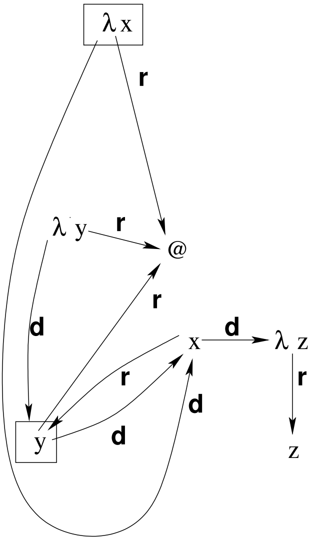


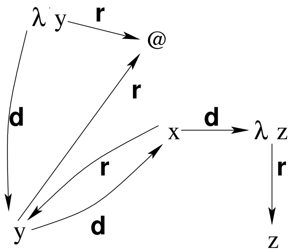


Joining Equivalent Nodes

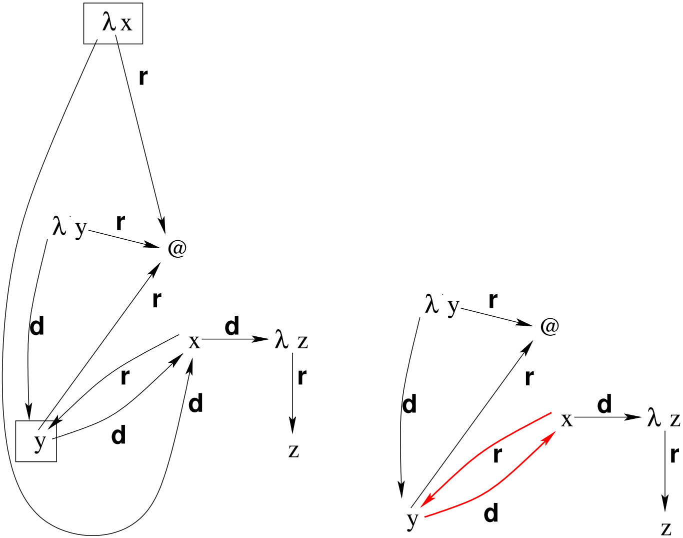


Joining Equivalent Nodes





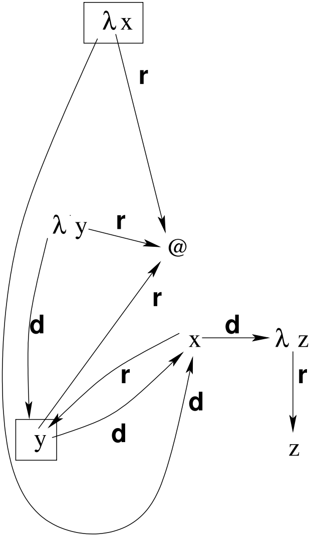
Joining Equivalent Nodes

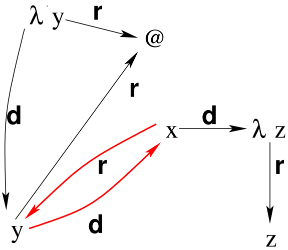


The graph is in normal form.

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

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.

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

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: $S \overset{X:=Y}{\longrightarrow_\eta} S'$, multiple steps are denoted by $S \overset{\nu}{\twoheadrightarrow_\eta} S'$ where ν is a substitution.
A scheme S is η-irreducible if no η-reduction steps are possible, the η-irreducible form of S is denoted by S_f.

Lemma

η-reduction is strongly normalising.

Definition

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

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, $S \overset{Z:=x}{\to_\eta} S'$, 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 $M \equiv f(fx)$.

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

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

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

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

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

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

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

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

Relation to the standard 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 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 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)	$\emptyset$	$\tau = \Gamma(x)$
$(\Gamma, \lambda x.M, \tau)$	$(\Gamma; x : \alpha_1, M, \alpha_2)$	$\tau = \alpha_1 \to \alpha_2$
$(\Gamma, M\,P, \tau)$	$(\Gamma, M, \alpha \to \tau), (\Gamma, P, \alpha)$	$\emptyset$

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

$$\mathrm{TVar} \doteq \mathrm{TVar} \to \mathrm{TVar}$$

Relation to the standard algorithm: Wand

Wand's original algorithm:

Action table:

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

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)	$\emptyset$	$\tau = \Gamma(x)$
$(\Gamma, \lambda x.M, \tau)$	$(\Gamma; x : \alpha_1, M, \alpha_2)$	$\tau = \alpha_1 \to \alpha_2$
$(\Gamma, M\,P, \tau)$	$(\Gamma, M, \alpha_1), (\Gamma, P, \alpha_2)$	$\alpha_1 = \alpha_2 \to \tau$

Generates a scheme of equations of the form

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

Relation to the standard algorithm
Scheme of type equations

Use Wand's algorithm to generate a scheme of equations of the following form

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

Example

The scheme $\mathcal{W}(M)$ of $M \equiv \underbrace{\lambda f^\alpha .\, \underbrace{\lambda x^\beta .\, \underbrace{f^\alpha(\overbrace{f^\alpha x^\beta}^{\delta})}_{\varepsilon}}_{\rho}}_{t}$ is

$$t \doteq \alpha \to \rho \qquad \rho \doteq \beta \to \varepsilon \qquad \alpha \doteq \delta \to \varepsilon \qquad \alpha \doteq \beta \to \delta$$

Relation to the standard algorithm

- Scheme of type equations (Wand) $\cong$ scheme of λ-term (Newman)
- Computation of most general unifier $\cong$ reduction of schemes

Relation to the standard algorithm

- Scheme of type equations (Wand) $\cong$ scheme of λ-term (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

Extension to weak polymorphism

Weak polymorphic λ-calculus

$$\mathsf{Type}_\omega ::= (\forall \mathsf{TVar}.\mathsf{Type}_\omega) \mid \mathsf{Type}$$

$$\frac{\Gamma, x : \sigma \vdash P : \tau}{\Gamma \vdash \lambda x.P : \sigma \to \tau}\; \sigma, \tau \in \mathsf{Type}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall \alpha.\sigma}\; \alpha \notin FTV(\Gamma)$$

$$\frac{\Gamma \vdash M : \forall \alpha.\sigma}{\Gamma \vdash M : \sigma[\alpha := \tau]}\; \tau \in \mathsf{Type}$$

Joining Equivalent Nodes

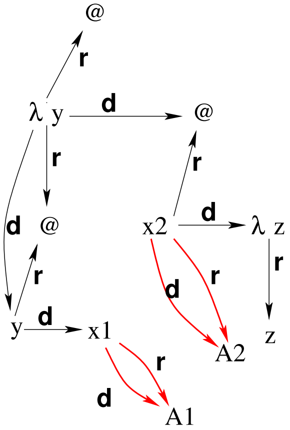
$\lambda x.\,(\lambda y.\, y\, x)\,(x\,(\lambda z.\, z))$ can be typed in weak polymorphic types as

$$x : \forall\alpha.\alpha \rightarrow \alpha \vdash (\lambda y.\, y\, x)\,(x\,(\lambda z.\, z)) :?$$

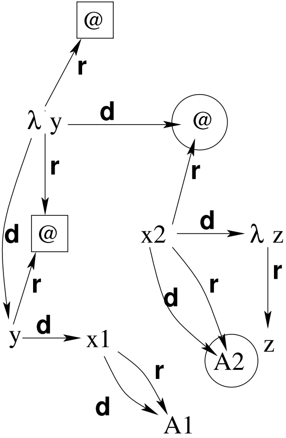
Joining Equivalent Nodes

$\lambda x.\,(\lambda y.\,y\,x)\,(x\,(\lambda z.\,z))$ can be typed in weak polymorphic types as

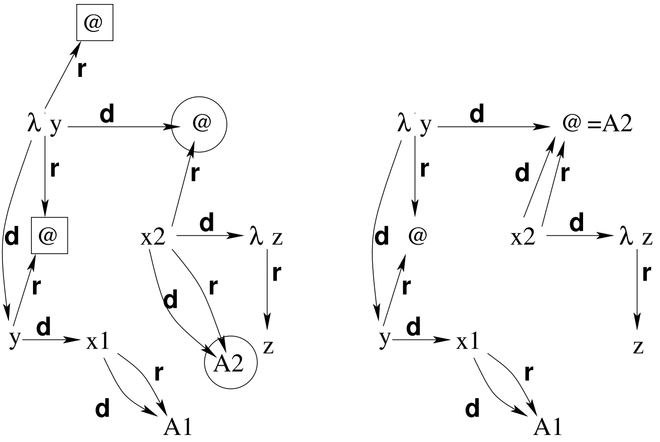
$$x : \forall\alpha.\alpha \to \alpha \vdash (\lambda y.\,y\,x)\,(x\,(\lambda z.\,z)) :?$$



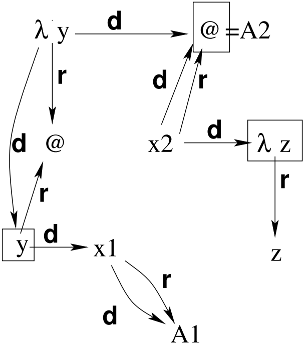
Joining Equivalent Nodes



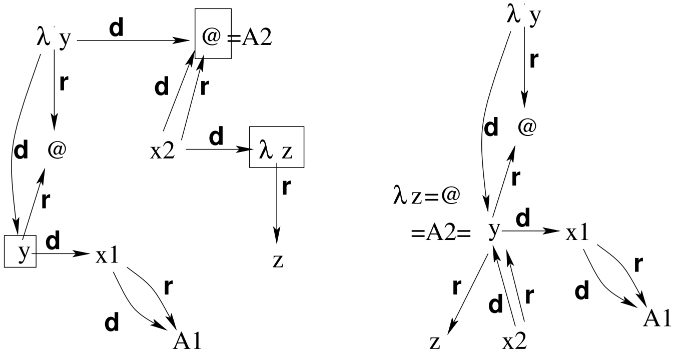
Joining Equivalent Nodes



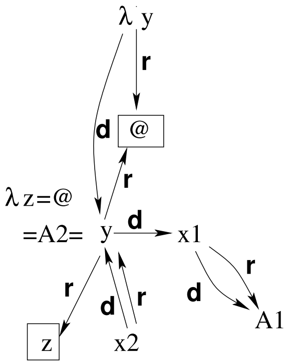
Joining Equivalent Nodes



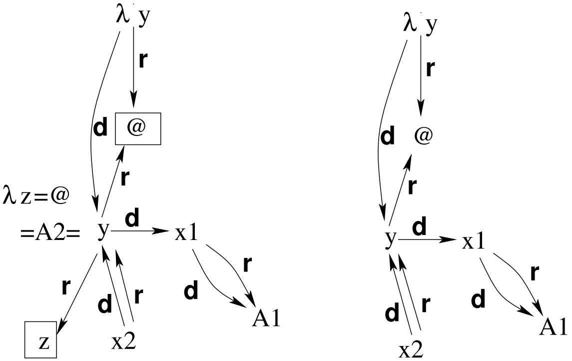
Joining Equivalent Nodes



Joining Equivalent Nodes



Joining Equivalent Nodes



The graph is in normal form.

It contains no cycle, so the term is typable.

Conclusions / Further research

- Not so different after all
- Original machinery quite heavy
- Nice tree representation of the algorithm
- Extend to *let* polymorphism?
- Extend to *dependent types*?

Questions

?