Type preservation in simply typed lambda calculus by abstract reduction techniques

Hans Zantema

Technische Universiteit Eindhoven and Radboud Universiteit Nijmegen
Joined work with Aaron Stump, Garrin Kimmell and Ruba El Haj Omar

TErm REwriting SEminar, December 9, 2011
Simply typed lambda calculus

types $T$ ::= $A \mid T_1 \Rightarrow T_2$

standard terms $t$ ::= $f \mid a \mid x \mid t_1 \ t_2 \mid \lambda x : T.\ t$

contexts $\Gamma$ ::= $\cdot \mid \Gamma, x : T$
Simply typed lambda calculus

**Types** $T$ ::=$A$ | $T_1 \Rightarrow T_2$

**Standard terms** $t$ ::=$f$ | $a$ | $x$ | $t_1 \ t_2$ | $\lambda x : T . t$

**Contexts** $\Gamma$ ::= $\cdot$ | $\Gamma, x : T$

**Typing rules:**

\[
\Gamma \vdash f : A \Rightarrow A \quad \Gamma \vdash a : A
\]

\[
\Gamma \vdash t_1 : T_2 \Rightarrow T_1 \quad \Gamma \vdash t_2 : T_2
\]

\[
\Gamma \vdash t_1 \ t_2 : T_1
\]

\[
\Gamma, x : T_1 \vdash t : T_2
\]

\[
\Gamma \vdash \lambda x : T_1 . t : T_1 \Rightarrow T_2
\]
Main theorem on type preservation:

\[(\Gamma \vdash t : T \land t \rightarrow t') \Rightarrow \Gamma \vdash t' : T\]
Main theorem on type preservation:

\[(\Gamma \vdash t : T \land t \rightarrow t') \Rightarrow \Gamma \vdash t' : T\]

For which computation steps \(\rightarrow?\)
Main theorem on type preservation:

\[(\Gamma \vdash t : T \wedge t \rightarrow t') \Rightarrow \Gamma \vdash t' : T\]

For which computation steps \(\rightarrow\)?

We do it for call-by-value: \(\beta\)-reduction steps in evaluation contexts:

- **values** \(v\) \(::=\) \(\lambda x : T.t\) | \(a\) | \(f\)
- **evaluation contexts** \(E\) \(::=\) \(\ast\) | \((E\ t)\) | \((v\ E)\)
Main theorem on type preservation:

\[ (\Gamma \vdash t : T \land t \rightarrow t') \Rightarrow \Gamma \vdash t' : T \]

For which computation steps →?

We do it for call-by-value: \( \beta \)-reduction steps in evaluation contexts:

Values:

\[
values \ v \ ::= \ \lambda x : T . t \mid a \mid f
\]

Evaluation contexts:

\[
evaluation \ contexts \ E \ ::= \ * \mid (E \ t) \mid (v \ E)
\]

\[
E[(\lambda x : T . t) \ v] \rightarrow E[[v/x]t]
\]

\[
E[f \ a] \rightarrow E[a]
\]
Type preservation

Main theorem on type preservation:

\[ (\Gamma \vdash t : T \land t \rightarrow t') \Rightarrow \Gamma \vdash t' : T \]

For which computation steps \( \rightarrow \)?

We do it for call-by-value: \( \beta \)-reduction steps in evaluation contexts:

\[
values \ v ::=} \ \lambda x : T . t \mid a \mid f \\
\]

\[
evaluation \ contexts \ E ::=} \ * \mid (E \ t) \mid (v \ E) \\
\]

\[
E[(\lambda x : T . t) \ v] \rightarrow E[[v/x]t] \\
E[f \ a] \rightarrow E[a]
\]

This is deterministic: if a term contains a \( \beta \)-redex, then exactly one call-by-value step is possible
The traditional proof is by induction on the structure of the typing derivation, with case analysis on the reduction derivation.
The traditional proof is by induction on the structure of the typing derivation, with case analysis on the reduction derivation.

A separate induction is required to prove a substitution lemma, needed critically for type preservation for $\beta$-reduction steps.
The traditional proof is by induction on the structure of the typing derivation, with case analysis on the reduction derivation.

A separate induction is required to prove a substitution lemma, needed critically for type preservation for $\beta$-reduction steps.

We give an alternative proof as a corollary of a general framework.
The traditional proof is by induction on the structure of the typing derivation, with case analysis on the reduction derivation.

A separate induction is required to prove a substitution lemma, needed critically for type preservation for $\beta$-reduction steps.

We give an alternative proof as a corollary of a general framework:

- we combine terms and types to a mixed syntax.
The traditional proof is by induction on the structure of the typing derivation, with case analysis on the reduction derivation.

A separate induction is required to prove a substitution lemma, needed critically for type preservation for $\beta$-reduction steps.

We give an alternative proof as a corollary of a general framework:

- we combine terms and types to a mixed syntax
- we allow both abstract steps (type computation) and concrete steps ($\beta$-reduction)
The traditional proof is by induction on the structure of the typing derivation, with case analysis on the reduction derivation.

A separate induction is required to prove a substitution lemma, needed critically for type preservation for $\beta$-reduction steps.

We give an alternative proof as a corollary of a general framework:

- we combine terms and types to a mixed syntax
- we allow both abstract steps (type computation) and concrete steps ($\beta$-reduction)
- we analyze for well-typed mixed terms how these abstract and concrete steps commute
The traditional proof is by induction on the structure of the typing derivation, with case analysis on the reduction derivation.

A separate induction is required to prove a substitution lemma, needed critically for type preservation for $\beta$-reduction steps.

We give an alternative proof as a corollary of a general framework:

- We combine terms and types to a mixed syntax.
- We allow both abstract steps (type computation) and concrete steps ($\beta$-reduction).
- We analyze for well-typed mixed terms how these abstract and concrete steps commute.
- These abstract properties imply type preservation.
Outline of the rest of the presentation

- Definition of mixed syntax
Outline of the rest of the presentation

- Definition of mixed syntax
- Definition of abstract steps (type computation) and concrete steps ($\beta$-reduction)
Outline of the rest of the presentation

- Definition of mixed syntax
- Definition of abstract steps (type computation) and concrete steps ($\beta$-reduction)
- Analysis of how these steps commute for well-typed mixed terms
Outline of the rest of the presentation

- Definition of mixed syntax
- Definition of abstract steps (type computation) and concrete steps ($\beta$-reduction)
- Analysis of how these steps commute for well-typed mixed terms
- Proof that this implies type preservation
Definition of mixed syntax

Definition of abstract steps (type computation) and concrete steps ($\beta$-reduction)

Analysis of how these steps commute for well-typed mixed terms

Proof that this implies type preservation

Proof that this implies confluence
Outline of the rest of the presentation

- Definition of mixed syntax
- Definition of abstract steps (type computation) and concrete steps ($\beta$-reduction)
- Analysis of how these steps commute for well-typed mixed terms
- Proof that this implies type preservation
- Proof that this implies confluence
- Technique to find counterexamples for possible generalizations of confluence theorem
Mixed syntax
Mixed syntax

types $T$ ::= $A \mid T_1 \Rightarrow T_2$

standard terms $t$ ::= $x \mid \lambda x : T . t \mid t \ t' \mid a \mid f$
Mixed syntax

types $T$ :: $A \mid T_1 \Rightarrow T_2$

standard terms $t$ :: $x \mid \lambda x : T. t \mid t \ t' \mid a \mid f$

mixed terms $m$ :: $x \mid \lambda x : T. m \mid m \ m' \mid a \mid f \mid A \mid T \Rightarrow m$
Mixed syntax

types $T$ ::= $A$ | $T_1 \Rightarrow T_2$

standard terms $t$ ::= $x$ | $\lambda x : T. t$ | $t \ t'$ | $a$ | $f$

mixed terms $m$ ::= $x$ | $\lambda x : T. m$ | $m \ m'$ | $a$ | $f$ | $A$ | $T \Rightarrow m$

standard values $v$ ::= $\lambda x : T. t$ | $a$ | $f$

mixed values $u$ ::= $\lambda x : T. m$ | $T \Rightarrow m$ | $A$ | $a$ | $f$
Mixed syntax

types $T$ ::= $A \mid T_1 \Rightarrow T_2$

standard terms $t$ ::= $x \mid \lambda x : T.t \mid t \ t' \mid a \mid f$

mixed terms $m$ ::= $x \mid \lambda x : T.m \mid m \ m' \mid a \mid f \mid A \mid T \Rightarrow m$

standard values $v$ ::= $\lambda x : T.t \mid a \mid f$

mixed values $u$ ::= $\lambda x : T.m \mid T \Rightarrow m \mid A \mid a \mid f$

Concrete reduction:

$$E_c[f \ a] \rightarrow_c E_c[a] \quad \quad \quad E_c[(\lambda x : T.m) \ u] \rightarrow_c E_c[[u/x]m]$$

concrete evaluation contexts $E_c ::= \ast \mid (E_c \ t) \mid (u \ E_c)$
Abstract reduction:

\[ E_a[(T \Rightarrow m) \; T] \rightarrow_a E_a[m] \]

\[ E_a[\lambda x : T. \; m] \rightarrow_a E_a[T \Rightarrow [T/x]m] \]

\[ E_a[f] \rightarrow_a E_a[A \Rightarrow A] \]

\[ E_a[a] \rightarrow_a E_a[A] \]
Abstract reduction:

\[
E_a[(T \Rightarrow m) \ T] \rightarrow_a E_a[m] \quad \quad \quad \quad \quad \quad E_a[\lambda x : T. \ m] \rightarrow_a E_a[T \Rightarrow [T/x]m]
\]

\[
E_a[f] \rightarrow_a E_a[A \Rightarrow A] \quad \quad \quad \quad \quad \quad E_a[a] \rightarrow_a E_a[A]
\]

Abstract evaluation contexts

\[
E_a ::= \ast \mid (E_a \ m) \mid (m \ E_a) \mid \lambda x : T. \ E_a \mid T \Rightarrow E_a
\]
Abstract reduction:

\[
E_a[(T \Rightarrow m) T] \rightarrow_a E_a[m] \quad E_a[\lambda x : T. m] \rightarrow_a E_a[T \Rightarrow [T/x]m]
\]
\[
E_a[f] \rightarrow_a E_a[A \Rightarrow A] \quad E_a[a] \rightarrow_a E_a[A]
\]

Abstract evaluation contexts

\[
E_a ::= * \mid (E_a \ m) \mid (m \ E_a) \mid \lambda x : T. E_a \mid T \Rightarrow E_a
\]

So abstract steps $\rightarrow_a$ for establishing types may be done in any context, concrete steps $\rightarrow_c$ describing real steps should follow the call-by-value format.
Example of typing by $\rightarrow_{a}$
Example of typing by $\rightarrow_a$

$$\lambda x : (A \Rightarrow A). \lambda y : A. (x (x y)) \rightarrow_a$$
$$\lambda x : (A \Rightarrow A). A \Rightarrow (x (x A)) \rightarrow_a$$
$$(A \Rightarrow A) \Rightarrow A \Rightarrow ((A \Rightarrow A) ((A \Rightarrow A) A)) \rightarrow_a$$
$$(A \Rightarrow A) \Rightarrow A \Rightarrow ((A \Rightarrow A) A) \rightarrow_a$$
$$(A \Rightarrow A) \Rightarrow A \Rightarrow A$$
Example of typing by $\rightarrow_a$

\[
\lambda x : (A \Rightarrow A). \lambda y : A. (x (x y)) \rightarrow_a \\
\lambda x : (A \Rightarrow A). A \Rightarrow (x (x A)) \rightarrow_a \\
(A \Rightarrow A) \Rightarrow A \Rightarrow ((A \Rightarrow A) ((A \Rightarrow A) A)) \rightarrow_a \\
(A \Rightarrow A) \Rightarrow A \Rightarrow ((A \Rightarrow A) A) \rightarrow_a \\
(A \Rightarrow A) \Rightarrow A \Rightarrow A
\]

**Theorem**

*For standard terms $t$ we have*

\[
x_1 : T_1, \cdots , x_n : T_n \vdash t : T
\]

*iff* $[T_1/x_1, \cdots , T_n/x_n]t \rightarrow^*_a T$
Theorem

If $m \rightarrow^*_a T$ and $m \rightarrow_c m'$, then $m' \rightarrow^*_a T$
Theorem

If $m \rightarrow^* a \land m \rightarrow_c m'$, then $m' \rightarrow^* a \land T$

For proving this we analyze local commutation of $\rightarrow_a$ and $\rightarrow_c$:

- If $m_1 \leftarrow_a m \rightarrow_a m_2$ for $m_1 \neq m_2$, then $m_3$ exists with $m_1 \rightarrow_a m_3 \leftarrow_a m_2$
- $\rightarrow_a$ has the diamond property
Generalized type preservation

**Theorem**

If \( m \rightarrow^*_{a} T \) and \( m \rightarrow_{c} m' \), then \( m' \rightarrow^*_{a} T \)

For proving this we analyze local commutation of \( \rightarrow_{a} \) and \( \rightarrow_{c} \):

- If \( m_1 \leftarrow_{a} m \rightarrow_{a} m_2 \) for \( m_1 \neq m_2 \), then \( m_3 \) exists with
  \[
  m_1 \rightarrow_{a} m_3 \leftarrow_{a} m_2
  \]
  \( (\rightarrow_{a} \) has the diamond property)\n
- \( m_1 \leftarrow_{c} m \rightarrow_{c} m_2 \) for \( m_1 \neq m_2 \) does not occur since \( \rightarrow_{c} \) is deterministic
Generalized type preservation

**Theorem**

If \( m \rightarrow_a^* T \) and \( m \rightarrow_c m' \), then \( m' \rightarrow_a^* T \)

For proving this we analyze local commutation of \( \rightarrow_a \) and \( \rightarrow_c \):

- If \( m_1 \leftarrow_a m \rightarrow_a m_2 \) for \( m_1 \neq m_2 \), then \( m_3 \) exists with
  \[ m_1 \rightarrow_a m_3 \leftarrow_a m_2 \]
  (\( \rightarrow_a \) has the diamond property)

- \( m_1 \leftarrow_c m \rightarrow_c m_2 \) for \( m_1 \neq m_2 \) does not occur since \( \rightarrow_c \) is deterministic

- If \( m_1 \leftarrow_a m \rightarrow_c m_2 \), then \( m_3 \) exists with either
  \[ m_1 \rightarrow_a^* m_3 \leftarrow_a^* m_2 \) or \( m_1 \rightarrow_c m_3 \leftarrow_a m_2 \)
Typical situation of last pattern:

\[
E_c[(\lambda x : T_1.m) u] \\
E_c[(T_1 \Rightarrow [T_1/x]m) u] \quad E_c[[u/x]m] \\
E_c[(T_1 \Rightarrow [T_1/x]m) T_1] \quad since \; u \rightarrow^*_a T_1 \\
E_c[[T_1/x]m] \\
since \; u \rightarrow^*_a T_1
\]
Theorem

If $m \rightarrow^*_a T$ and $m \rightarrow_c m'$, then $m' \rightarrow^*_a T$
Theorem

If \( m \rightarrow_{a}^{\ast} T \) and \( m \rightarrow_{c} m' \), then \( m' \rightarrow_{a}^{\ast} T \)

This theorem is easily proved by induction on the length of \( m \rightarrow_{a}^{\ast} T \), using the local commutation properties
Theorem

If \( m \rightarrow_{a}^{*} T \) and \( m \rightarrow_{c} m' \), then \( m' \rightarrow_{a}^{*} T \)

This theorem is easily proved by induction on the length of \( m \rightarrow_{a}^{*} T \), using the local commutation properties.

Restricted to standard terms this theorem implies type preservation of call-by-value steps:
Theorem

If \( m \xrightarrow{\ast}_a T \) and \( m \xrightarrow{c} m' \), then \( m' \xrightarrow{\ast}_a T \)

This theorem is easily proved by induction on the length of \( m \xrightarrow{\ast}_a T \), using the local commutation properties.

Restricted to standard terms this theorem implies type preservation of call-by-value steps:

If \( m \) is a typable term of type \( T \) and \( m \xrightarrow{c} m' \), then \( m \xrightarrow{\ast}_a T \)
Theorem

If $m \rightarrow^*_a T$ and $m \rightarrow_c m'$, then $m' \rightarrow^*_a T$

This theorem is easily proved by induction on the length of $m \rightarrow^*_a T$, using the local commutation properties.

Restricted to standard terms this theorem implies type preservation of call-by-value steps:

if $m$ is a typable term of type $T$ and $m \rightarrow_c m'$, then $m \rightarrow^*_a T$

Theorem $\Rightarrow m' \rightarrow^*_a T$, so $m'$ is typable with type $T$
Theorem

If $m \xrightarrow{\ast}_a T$ then $m$ is confluent wrt $\rightarrow = \rightarrow_a \cup \rightarrow_c$
Confluence

Theorem

If $m \rightarrow^{*}_a T$ then $m$ is confluent wrt $\rightarrow = \rightarrow_a \cup \rightarrow_c$

Here an element $m$ is called *confluent* wrt to $\rightarrow$ if for all $x, y$ satisfying

$$x \leftarrow^{*} m \rightarrow^{*} y$$

an element $z$ exists such that

$$x \rightarrow^{*} z \leftarrow^{*} y$$
Confluence

**Theorem**

If \( m \rightarrow^*_a T \) then \( m \) is confluent wrt \( \rightarrow = \rightarrow_a \cup \rightarrow_c \)

Here an element \( m \) is called *confluent* wrt to \( \rightarrow \) if for all \( x, y \) satisfying

\[
x \leftarrow^* m \rightarrow^* y
\]

an element \( z \) exists such that

\[
x \rightarrow^* z \leftarrow^* y
\]

The typability requirement \( m \rightarrow^*_a T \) is essential: the non-typable term \( (\lambda x : A.x)(\lambda x : A.x) \) has two distinct normal forms

\[
(A \Rightarrow A)(A \Rightarrow A) \leftarrow^+_a (\lambda x : A.x)(\lambda x : A.x) \rightarrow^+_c \lambda x : A.x \rightarrow_a (A \Rightarrow A)
\]
Using the local commutation properties we already observed for \( \rightarrow_a \) and \( \rightarrow_c \) we have to prove
Using the local commutation properties we already observed for $\rightarrow_a$ and $\rightarrow_c$ we have to prove

**Theorem**

*Let $\rightarrow_a$ and $\rightarrow_c$ be relations such that*

- $\rightarrow_a$ has the diamond property,
- $\rightarrow_c$ is deterministic, and

$$\leftarrow_a \cdot \rightarrow_c \subseteq (\rightarrow_a^* \cdot \leftarrow_a^*) \cup (\rightarrow_c \cdot \leftarrow_a)$$

*Then $CR(\rightarrow_a \cup \rightarrow_c)$*
Using the local commutation properties we already observed for $\rightarrow_a$ and $\rightarrow_c$ we have to prove

**Theorem**

Let $\rightarrow_a$ and $\rightarrow_c$ be relations such that

- $\rightarrow_a$ has the diamond property,
- $\rightarrow_c$ is deterministic, and
- $\leftarrow_a \cdot \rightarrow_c \subseteq (\rightarrow_a^* \cdot \leftarrow_a^*) \cup (\rightarrow_c \cdot \leftarrow_a)$

Then $\text{CR}(\rightarrow_a \cup \rightarrow_c)$

Indeed this theorem holds
For variants of the theory (simply typed combinators with uniform syntax) we need a generization, in particular, $\rightarrow_a$ is confluent but does not have the diamond property any more.
For variants of the theory (simply typed combinators with uniform syntax) we need a generization, in particular, $\to_a$ is confluent but does not have the diamond property any more.

Our goal is to find and prove a generalization of the above theorem on abstract reduction to conclude confluence of the union of two basic relations $\to_1$ and $\to_2$.
For variants of the theory (simply typed combinators with uniform syntax) we need a generalization, in particular, $\rightarrow_a$ is confluent but does not have the diamond property any more.

Our goal is to find and prove a generalization of the above theorem on abstract reduction to conclude confluence of the union of two basic relations $\rightarrow_1$ and $\rightarrow_2$.

The strongest version we found:
For variants of the theory (simply typed combinators with uniform syntax) we need a generization, in particular, $\rightarrow_a$ is confluent but does not have the diamond property any more.

Our goal is to find and prove a generalization of the above theorem on abstract reduction to conclude confluence of the union of two basic relations $\rightarrow_1$ and $\rightarrow_2$.

The strongest version we found:

**Theorem**

Let $\rightarrow_1$ and $\rightarrow_2$ be relations such that

- $CR(\rightarrow_1)$,
- $\leftarrow_2 \cdot \rightarrow_2 \subseteq \rightarrow_1^* \cdot \rightarrow_2^* \cdot \rightarrow_1^* \cdot \leftarrow_2^* \cdot \leftarrow_1^*$,
- $\leftarrow_1 \cdot \rightarrow_2 \subseteq \rightarrow_2^* \cdot \rightarrow_1^* \cdot \leftarrow_1^*$.

Then $CR(\rightarrow_1 \cup \rightarrow_2)$.
Two proofs
(1) Apply Decreasing Diagram Theorem [VvO 1994] to $\rightarrow_1^*$ and $\rightarrow_2$
Two proofs

(1) Apply Decreasing Diagram Theorem [VvO 1994] to $\rightarrow_1^*$ and $\rightarrow_2$

(2) A direct proof using termination of

\[
\begin{align*}
  aA & \rightarrow Aa \\
  bB & \rightarrow ABAaba \\
  aB & \rightarrow BAa \\
  bA & \rightarrow Aab \\
  b & \rightarrow \epsilon \\
  B & \rightarrow \epsilon
\end{align*}
\]

\[
b = \leftarrow_2, \quad B = \rightarrow_2, \quad a = \leftarrow_1^*, \quad A = \rightarrow_1^*
\]
Requirement $\leftarrow_2 \cdot \rightarrow_2 \subseteq \cdots$ may NOT be replaced by $CR(\rightarrow_2)$
Requirement $\leftarrow_2 \cdot \rightarrow_2 \subseteq \cdots$ may NOT be replaced by $CR(\rightarrow_2)$

Example:

$\rightarrow_1$ steps are denoted by dashed arrows
$\rightarrow_2$ steps are denoted by solid arrows
Not even if moreover both $\rightarrow_1$ and $\rightarrow_2$ are required to be terminating
Not even if moreover both $\rightarrow_1$ and $\rightarrow_2$ are required to be terminating

Example:

$\rightarrow_1$: dashed arrows, $\rightarrow_2$: solid arrows
How were these examples found?
How were these examples found?

By SAT solving
How were these examples found?

By SAT solving

Fix a number $n$
How were these examples found?

By SAT solving

Fix a number $n$

For every binary relation $R$ introduce $n^2$ boolean variables $R_{ij}$ indicating whether $(i, j)$ is in the relation or not.
How were these examples found?

By SAT solving

Fix a number $n$

For every binary relation $R$ introduce $n^2$ boolean variables $R_{ij}$ indicating whether $(i, j)$ is in the relation or not

Express all conditions and the negation of the conclusion by a set of boolean constraints
How were these examples found?

By SAT solving

Fix a number $n$

For every binary relation $R$ introduce $n^2$ boolean variables $R_{ij}$ indicating whether $(i, j)$ is in the relation or not

Express all conditions and the negation of the conclusion by a set of boolean constraints

This may require several auxiliary relations, e.g., for $n = 8$ the relation $R^*$ may be defined by

$$R_2 = I \cup R \cup R; \quad R_4 = R_2; \quad R_2, \quad R^* = R_4; R_4$$
How were these examples found?

By SAT solving

Fix a number $n$

For every binary relation $R$ introduce $n^2$ boolean variables $R_{ij}$ indicating whether $(i,j)$ is in the relation or not

Express all conditions and the negation of the conclusion by a set of boolean constraints

This may require several auxiliary relations, e.g., for $n = 8$ the relation $R^*$ may be defined by

$$R_2 = I \cup R \cup R; R, \quad R_4 = R_2; R_2, \quad R^* = R_4; R_4$$

For the first example a lot of symmetry was observed, for the second example this symmetry was added as extra requirement
Conclusions

- We introduced a mixed calculus both containing type calculation and \textit{beta}-reduction
Conclusions

- We introduced a mixed calculus both containing type calculation and \textit{beta}-reduction.

- In this mixed calculus we defined $\rightarrow_a$ steps for type calculation and $\rightarrow_c$ representing call-by-value steps.
Conclusions

- We introduced a mixed calculus both containing type calculation and \emph{beta}-reduction.

- In this mixed calculus we defined $\rightarrow_a$ steps for type calculation and $\rightarrow_c$ representing call-by-value steps.

- Local commutation of $\rightarrow_a$ steps and $\rightarrow_c$ steps was investigated.
Conclusions

- We introduced a mixed calculus both containing type calculation and beta-reduction.
- In this mixed calculus we defined \( \rightarrow_a \) steps for type calculation and \( \rightarrow_c \) representing call-by-value steps.
- Local commutation of \( \rightarrow_a \) steps and \( \rightarrow_c \) steps was investigated.
- Two results: type preservation and confluence for typable terms.
Conclusions

- We introduced a mixed calculus both containing type calculation and beta-reduction.
- In this mixed calculus we defined $\rightarrow_a$ steps for type calculation and $\rightarrow_c$ representing call-by-value steps.
- Local commutation of $\rightarrow_a$ steps and $\rightarrow_c$ steps was investigated.
- Two results: type preservation and confluence for typable terms.
- For confluence we developed a theorem on abstract reduction.
Conclusions

- We introduced a mixed calculus both containing type calculation and \( \beta \)-reduction.

- In this mixed calculus we defined \( \rightarrow_a \) steps for type calculation and \( \rightarrow_c \) representing call-by-value steps.

- Local commutation of \( \rightarrow_a \) steps and \( \rightarrow_c \) steps was investigated.

- Two results: type preservation and confluence for typable terms.

- For confluence we developed a theorem on abstract reduction.

- Possible generalizations were violated by finding counter examples using SAT solving.
Conclusions

- We introduced a mixed calculus both containing type calculation and \textit{beta}-reduction.
- In this mixed calculus we defined \( \rightarrow_a \) steps for type calculation and \( \rightarrow_c \) representing call-by-value steps.
- Local commutation of \( \rightarrow_a \) steps and \( \rightarrow_c \) steps was investigated.
- Two results: type preservation and confluence for typable terms.
- For confluence we developed a theorem on abstract reduction.
- Possible generalizations were violated by finding counter examples using SAT solving.
- Desirable extension: arbitrary \( \beta \)-steps rather than call-by-value.