Daria Walukiewicz-Chrząszcz, Jacek Chrząszcz

Inductive Consequences in the Calculus of Constructions

Institute of Informatics
University of Warsaw

Conversion in CIC

CC + Inductive Definitions = Calculus of Inductive Constructions

conversion rule:

(conv) $$\frac{E \vdash e : t \qquad E \vdash t \stackrel{\star}{\longleftrightarrow} t'}{E \vdash e : t'}$$

$$\longrightarrow \; = \; \longrightarrow_\beta \cup \longrightarrow_\iota$$

Conversion in CIC

CC + Inductive Definitions = Calculus of Inductive Constructions

conversion rule:

$$\textbf{(conv)} \qquad \frac{E \vdash e : t \qquad E \vdash t \stackrel{\star}{\longleftrightarrow} t'}{E \vdash e : t'}$$

$$\longrightarrow \ = \ \longrightarrow_{\beta} \cup \longrightarrow_{\iota}$$

Larger conversion results in simpler and smaller proofs

Adding rewriting

`Symbol + : nat` $\longrightarrow$ `nat` $\longrightarrow$ `nat`

Adding rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l).
```

Adding rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l).

Axiom fact : P (0 + (m + 0)).
```

Adding rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l).

Axiom fact : P (0 + (m + 0)).

Goal (P m).
  exact fact.
Qed.
```

Adding rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l).

Axiom fact : P (0 + (m + 0)).

Goal (P m).
  exact fact.
Qed.
```

$$\textbf{(conv)}\qquad \frac{E \vdash e : t \qquad E \vdash t \overset{\star}{\longleftrightarrow} t'}{E \vdash e : t'}$$

$$\longrightarrow \;=\; \longrightarrow_\beta \cup \longrightarrow_R$$

Rewriting in the calculus of constructions

- rewriting packages in environments

$$E;\ \mathsf{Rew}(\, f\ :\ t;\ \{\ldots f\,\vec{l} \longrightarrow r\ldots\}\,);\ E'$$

Rewriting in the calculus of constructions

- rewriting packages in environments

$$E;\ \mathsf{Rew}(\ f\ :\ t;\ \{\ldots f\,\vec{l} \longrightarrow r \ldots\}\ );\ E'$$

- conversion depends on the environment

Rewriting in the calculus of constructions

- rewriting packages in environments

$$E;\ \mathsf{Rew}(\,f\ :\ t;\ \{\ldots f\,\vec{l} \longrightarrow r \ldots\}\,);\ E'$$

- conversion depends on the environment

- f can be higher order, polymorphic, dependent

Rewriting in the calculus of constructions

- rewriting packages in environments

$$E;\ \mathsf{Rew}(\, f\ :\ t;\ \{\ldots f\,\vec{l} \longrightarrow r \ldots\}\,);\ E'$$

- conversion depends on the environment

- f can be higher order, polymorphic, dependent

- syntactic matching

$$e[l\sigma]_p \longrightarrow_R e[r\sigma]_p$$

Rewriting in the calculus of constructions

- rewriting packages in environments

$$E;\ \mathsf{Rew}(\ f\ :\ t;\ \{\dots f\,\vec{l} \longrightarrow r \dots\}\ );\ E'$$

- conversion depends on the environment

- f can be higher order, polymorphic, dependent

- syntactic matching

$$e[l\sigma]_p \longrightarrow_R e[r\sigma]_p$$

CC+Rew = CC + Ind types + Rew packages + new conversion

Rewriting in the calculus of constructions

- rewriting packages in environments

$$E;\ \mathsf{Rew}(\, f\ :\ t;\ \{\ldots f\,\vec{l} \longrightarrow r \ldots\}\,);\ E'$$

- conversion depends on the environment
- f can be higher order, polymorphic, dependent
- syntactic matching

$$e[l\sigma]_p \longrightarrow_R e[r\sigma]_p$$

CC+Rew = CC + Ind types + Rew packages + new conversion

$$\textbf{(conv)} \quad \frac{E \vdash e : t \qquad E \vdash t \overset{\star}{\longleftrightarrow} t'}{E \vdash e : t'}$$

$$\longrightarrow\ =\ \longrightarrow_\beta \cup \longrightarrow_R$$

Known metatheoretical properties of CC+Rew

- termination of $\longrightarrow_R \cup \longrightarrow_\beta$
  General Schema (F.Blanqui, PhD 2001),
  HORPO (D.Walukiewicz, PhD 2003)
- confluence: (F.Müler, 1992)
- subject reduction

Known metatheoretical properties of CC+Rew

- termination of $\longrightarrow_R \cup \longrightarrow_\beta$
  General Schema (F.Blanqui, PhD 2001),
  HORPO (D.Walukiewicz, PhD 2003)
- confluence: (F.Müler, 1992)
- subject reduction
- logical consistency,

Known metatheoretical properties of CC+Rew

- termination of $\longrightarrow_R \cup \longrightarrow_\beta$
  General Schema (F.Blanqui, PhD 2001),
  HORPO (D.Walukiewicz, PhD 2003)
- confluence: (F.Müler, 1992)
- subject reduction
- logical consistency, completeness of definitions by rewriting
  (D.Walukiewicz & J.Ch., IJCAR 2006, LMCS 2008)

Known metatheoretical properties of CC+Rew

- termination of $\longrightarrow_R \cup \longrightarrow_\beta$
  General Schema (F.Blanqui, PhD 2001),
  HORPO (D.Walukiewicz, PhD 2003)
- confluence: (F.Müler, 1992)
- subject reduction
- logical consistency, completeness of definitions by rewriting
  (D.Walukiewicz & J.Ch., IJCAR 2006, LMCS 2008)
- missing: logical power of CC+Rew,

Known metatheoretical properties of CC+Rew

- termination of $\longrightarrow_R \cup \longrightarrow_\beta$
  General Schema (F.Blanqui, PhD 2001),
  HORPO (D.Walukiewicz, PhD 2003)
- confluence: (F.Müler, 1992)
- subject reduction
- logical consistency, completeness of definitions by rewriting
  (D.Walukiewicz & J.Ch., IJCAR 2006, LMCS 2008)
- missing: logical power of CC+Rew,(conservativity ?)

Complete definitions by rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l)
```

Complete definitions by rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l)
```

```
Symbol head : ∀ k:nat, vector (S k) → bool
```

Complete definitions by rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l)
```

```
Symbol head : ∀ k:nat, vector (S k) → bool
Rules head k (vcons a k l) ⟶ a
```

Complete definitions by rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l)
```

```
Symbol head : ∀ k:nat, vector (S k) → bool
Rules head k (vcons a k l) ⟶ a
```

```
Symbol K : ∀ (A:Set) (a:A) (P:eq A a a → Set),
           P (refl A a) → ∀ p: eq A a a, P p
```

Complete definitions by rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l)

Symbol head : ∀ k:nat, vector (S k) → bool
Rules head k (vcons a k l) ⟶ a

Symbol K : ∀ (A:Set) (a:A) (P:eq A a a → Set),
           P (refl A a) → ∀ p: eq A a a, P p
Rules K A a P h (refl A a) ⟶ h
```

Complete definitions by rewriting

```
Symbol + : nat → nat → nat
Rules 0 + m ⟶ m
      (S n) + m ⟶ S (n + m)        ⟸ complete
      n + 0 ⟶ n
      n + (S m) ⟶ S (n + m)
      (n + m) + l ⟶ n + (m + l)
```

```
Symbol head : ∀ k:nat, vector (S k) → bool
Rules head k (vcons a k l) ⟶ a
```

```
Symbol K : ∀ (A:Set) (a:A) (P:eq A a a → Set),
           P (refl A a) → ∀ p: eq A a a, P p
Rules K A a P h (refl A a) ⟶ h
```

Complete subsystem and other rules

$$R' \begin{cases} R \begin{cases} f\ \vec{l_1} \longrightarrow r_1 \\ f\ \vec{l_2} \longrightarrow r_2 \end{cases} \\ f\ \vec{l_3} \longrightarrow r_3 \\ f\ \vec{l_4} \longrightarrow r_4 \\ f\ \vec{l_5} \longrightarrow r_5 \end{cases}$$

complete (often a pattern-matching definition of f)

Complete subsystem and other rules

$$R' \begin{cases} R \begin{cases} f\ \vec{l_1} \longrightarrow r_1 \\ f\ \vec{l_2} \longrightarrow r_2 \end{cases} \\ f\ \vec{l_3} \longrightarrow r_3 \\ f\ \vec{l_4} \longrightarrow r_4 \\ f\ \vec{l_5} \longrightarrow r_5 \end{cases}$$

complete (often a pattern-matching definition of f)

Question

How are the rules in $R' \setminus R$ related to R ?

First order case

Other rules are inductive consequences:

Lemma

If $R \subseteq R'$ and they are both terminating, confluent and R is complete, then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \stackrel{\star}{\longleftrightarrow}_R r\sigma$.

First order case

Other rules are inductive consequences:

Lemma

If $R \subseteq R'$ and they are both terminating, confluent and R is complete, then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \overset{\star}{\longleftrightarrow}_R r\sigma$.

(conclusion from an old and simple theorem)

Our contribution

Our contribution

Extension of inductive consequences lemma to higher-order rewriting on:

- non-functional inductive types
- functional inductive types

Inductive consequences for **non-functional** types

Inductive consequences for non-functional types

Theorem 1

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and critical pairs are joinable without $\longrightarrow_{R' \setminus R}$ under a binder, then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \stackrel{\star}{\longleftrightarrow}_R r\sigma$.

Inductive consequences for non-functional types

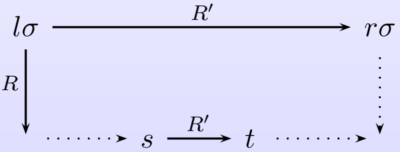
Theorem 1

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and critical pairs are joinable without $\longrightarrow_{R' \setminus R}$ under a binder, then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \overset{\star}{\longleftrightarrow}_R r\sigma$.

Inductive consequences for non-functional types

Theorem 1

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and critical pairs are joinable without $\longrightarrow_{R' \setminus R}$ under a binder, then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \overset{\star}{\longleftrightarrow}_R r\sigma$.



$$s[l'\sigma']_p \xrightarrow{R'} s[r'\sigma']_p$$

by induction on $\longrightarrow_{R'} \cup \longrightarrow_\beta \cup \rhd$

Inductive consequences for non-functional types

```
Inductive list : Set := nil : list | cons : bool→list→list
```

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

`Rules`

Inductive consequences for non-functional types

```
Inductive list : Set := nil : list | cons : bool→list→list

Symbol map : (bool→bool) → list → list
Rules       map F nil ⟶ nil
            map F (cons a l) ⟶ cons (F a) (map F l)
```

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

`Rules` map F nil $\longrightarrow$ nil

map F (cons a l) $\longrightarrow$ cons (F a) (map F l)

map $\lambda x.x$ l $\longrightarrow$ l

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Rules

$$R: \begin{array}{l} \texttt{map F nil} \longrightarrow \texttt{nil} \\ \texttt{map F (cons a l)} \longrightarrow \texttt{cons (F a) (map F l)} \\ \texttt{map } \lambda x.x \texttt{ l} \longrightarrow \texttt{l} \end{array}$$

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Rules

$$R : \begin{array}{l} \texttt{map F nil} \longrightarrow \texttt{nil} \\ \texttt{map F (cons a l)} \longrightarrow \texttt{cons (F a) (map F l)} \end{array}$$

$$R' \setminus R \ni \texttt{map } \lambda x.x \texttt{ l} \longrightarrow \texttt{l}$$

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Rules

R : `map F nil` $\longrightarrow$ `nil`

`map F (cons a l)` $\longrightarrow$ `cons (F a) (map F l)`

$R' \setminus R \ni$ `map` $\lambda x.x$ `l` $\longrightarrow$ `l`

`map` $\lambda x.x$ `nil`

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

`Rules`

R : `map F nil` $\longrightarrow$ `nil`

`map F (cons a l)` $\longrightarrow$ `cons (F a) (map F l)`

$R' \setminus R \ni$ `map` $\lambda x.x$ `l` $\longrightarrow$ `l`

`map` $\lambda x.x$ `nil` $\longrightarrow_{R'}$ `nil`

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Rules

R : `map F nil` $\longrightarrow$ `nil`

`map F (cons a l)` $\longrightarrow$ `cons (F a) (map F l)`

$R' \setminus R \ni$ `map` $\lambda x.x$ `l` $\longrightarrow$ `l`

`map` $\lambda x.x$ `nil` $\longrightarrow_{R'}$ `nil`

$\downarrow_R$

`nil`

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Rules

R : `map F nil` $\longrightarrow$ `nil`

`map F (cons a l)` $\longrightarrow$ `cons (F a) (map F l)`

$R' \setminus R \ni$ `map` $\lambda x.x$ `l` $\longrightarrow$ `l`

`map` $\lambda x.x$ `nil` $\longrightarrow_{R'}$ `nil`

$\downarrow_R$

`nil`

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Rules
$$R : \begin{array}{l}\texttt{map F nil} \longrightarrow \texttt{nil} \\ \texttt{map F (cons a l)} \longrightarrow \texttt{cons (F a) (map F l)}\end{array}$$
$$R' \setminus R \ni \texttt{map } \lambda x.x \texttt{ l} \longrightarrow \texttt{l}$$

$$\begin{array}{l}\texttt{map } \lambda x.x \texttt{ nil} \longrightarrow_{R'} \texttt{nil} \\ \downarrow_R \\ \texttt{nil}\end{array}$$

$\texttt{map } \lambda x.x \texttt{ (cons a l)}$

Inductive consequences for non-functional types

Inductive list : Set := nil : list | cons : bool→list→list

Symbol map : (bool→bool) → list → list

Rules

R : map F nil ⟶ nil

map F (cons a l) ⟶ cons (F a) (map F l)

$R' \setminus R \ni$ map $\lambda x.x$ l ⟶ l

map $\lambda x.x$ nil $\longrightarrow_{R'}$ nil

$\downarrow_R$

nil

map $\lambda x.x$ (cons a l) $\longrightarrow_{R'}$ cons a l

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Rules

$$R : \begin{array}{l} \texttt{map F nil} \longrightarrow \texttt{nil} \\ \texttt{map F (cons a l)} \longrightarrow \texttt{cons (F a) (map F l)} \end{array}$$

$$R' \setminus R \ni \texttt{map } \lambda x.x \texttt{ l} \longrightarrow \texttt{l}$$

$$\begin{array}{lll} \texttt{map } \lambda x.x \texttt{ nil} & \longrightarrow_{R'} & \texttt{nil} \\ \downarrow_R & & \\ \texttt{nil} & & \end{array}$$

$$\begin{array}{lll} \texttt{map } \lambda x.x \texttt{ (cons a l)} & \longrightarrow_{R'} & \texttt{cons a l} \\ \downarrow_R & & \\ \texttt{cons } (\lambda x.x \texttt{ a) (map } \lambda x.x \texttt{ l)} & & \end{array}$$

Inductive consequences for non-functional types

```
Inductive list : Set := nil : list | cons : bool→list→list
```

```
Symbol map : (bool→bool) → list → list
```

Rules $\quad R :$ `map F nil` $\longrightarrow$ `nil`

`map F (cons a l)` $\longrightarrow$ `cons (F a) (map F l)`

$R' \setminus R \ni$ `map` $\lambda x.x$ `l` $\longrightarrow$ `l`

`map` $\lambda x.x$ `nil` $\longrightarrow_{R'}$ `nil`

$\downarrow_R$

`nil`

`map` $\lambda x.x$ `(cons a l)` $\longrightarrow_{R'}$ `cons a l`

$\downarrow_R$

`cons (`$\lambda x.x$ `a) (map` $\lambda x.x$ `l)` $\longrightarrow_\beta$ `cons a (map` $\lambda x.x$ `l)`

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Rules $\quad R :$ `map F nil` $\longrightarrow$ `nil`

`map F (cons a l)` $\longrightarrow$ `cons (F a) (map F l)`

$R' \setminus R \ni$ `map` $\lambda x.x$ `l` $\longrightarrow$ `l`

$$\texttt{map}\ \lambda x.x\ \texttt{nil} \longrightarrow_{R'} \texttt{nil}$$
$$\downarrow_R$$
$$\texttt{nil}$$

$$\texttt{map}\ \lambda x.x\ (\texttt{cons a l}) \longrightarrow_{R'} \texttt{cons a l}$$
$$\downarrow_R \qquad\qquad \uparrow^{R'}$$
$$\texttt{cons}\ (\lambda x.x\ \texttt{a})\ (\texttt{map}\ \lambda x.x\ \texttt{l}) \longrightarrow_{\beta} \texttt{cons a}\ (\texttt{map}\ \lambda x.x\ \texttt{l})$$

Inductive consequences for non-functional types

`Inductive list : Set := nil : list | cons : bool→list→list`

`Symbol map : (bool→bool) → list → list`

Rules

$$R: \begin{array}{l} \texttt{map F nil} \longrightarrow \texttt{nil} \\ \texttt{map F (cons a l)} \longrightarrow \texttt{cons (F a) (map F l)} \end{array}$$

$$R' \setminus R \ni \texttt{map}\ \lambda x.x\ \texttt{l} \longrightarrow \texttt{l}$$

$$\begin{array}{lll} \texttt{map}\ \lambda x.x\ \texttt{nil} & \longrightarrow_{R'} & \texttt{nil} \\ \downarrow_R & & \\ \texttt{nil} & & \end{array}$$

$$\begin{array}{lll} \texttt{map}\ \lambda x.x\ \texttt{(cons a l)} & \longrightarrow_{R'} & \texttt{cons a l} \\ \downarrow_R & & \uparrow^{R'} \\ \texttt{cons}\ (\lambda x.x\ \texttt{a})\ (\texttt{map}\ \lambda x.x\ \texttt{l}) & \longrightarrow_\beta & \texttt{cons a (map}\ \lambda x.x\ \texttt{l)} \end{array}$$

Inductive consequences for non-functional types

Theorem 1

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and critical pairs are joinable without $\longrightarrow_{R' \setminus R}$ under a binder,
then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \stackrel{\star}{\longleftrightarrow}_R r\sigma$.

Inductive consequences for non-functional types

Theorem 1

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and critical pairs are joinable without $\longrightarrow_{R' \setminus R}$ under a binder, then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \stackrel{\star}{\longleftrightarrow}_R r\sigma$.

Inductive consequences for **functional** types

Inductive consequences for functional types

```
Inductive ord : Set :=
| o : ord
| s : ord → ord
| lim : (nat → ord) → ord.
```

Inductive consequences for functional types

```
Inductive ord : Set :=
| o : ord
| s : ord → ord
| lim : (nat → ord) → ord.
```

```
Symbol id : ord → ord
Rules id o ⟶ o
      id (s x) ⟶ s (id x)
      id (lim F) ⟶ lim (λ n. id (F n))
```

Inductive consequences for functional types

```
Inductive ord : Set :=
| o : ord
| s : ord → ord
| lim : (nat → ord) → ord.
```

```
Symbol id : ord → ord
Rules id o ⟶ o
      id (s x) ⟶ s (id x)
      id (lim F) ⟶ lim (λ n. id (F n))

      id (id x) ⟶ id x
```

Inductive consequences with functional inductive types

$$R \ni \texttt{id (lim F)} \longrightarrow \texttt{lim (}\lambda\texttt{ n. id (F n))}$$
$$R' \setminus R \ni \texttt{id (id x)} \longrightarrow \texttt{id x}$$

Inductive consequences with functional inductive types

$$R \ni \texttt{id (lim F)} \longrightarrow \texttt{lim } (\lambda \texttt{ n. id (F n))}$$
$$R' \setminus R \ni \texttt{id (id x)} \longrightarrow \texttt{id x}$$

$$\texttt{id (id (lim F))} \longrightarrow_{R'} \texttt{id (lim F)}$$
$$\downarrow_R$$
$$\texttt{id (lim } (\lambda \texttt{ n'. id (F n')))}$$

Inductive consequences with functional inductive types

$$R \ni \texttt{id (lim F)} \longrightarrow \texttt{lim (}\lambda\texttt{ n. id (F n))}$$

$$R' \setminus R \ni \texttt{id (id x)} \longrightarrow \texttt{id x}$$

$$\texttt{id (id (lim F))} \longrightarrow_{R'} \texttt{id (lim F)} \longrightarrow_{R} \texttt{lim (}\lambda\texttt{ n'.id (F n'))}$$

$$\downarrow_R$$

$$\texttt{id (lim (}\lambda\texttt{ n'. id (F n')))}$$

$$\downarrow_R$$

$$\texttt{lim (}\lambda\texttt{ n''. id ((}\lambda\texttt{ n'. id (F n')) n''))}$$

$$\downarrow_\beta$$

$$\texttt{lim (}\lambda\texttt{ n''. id (id (F n'')))} \longrightarrow_{R'} \texttt{lim (}\lambda\texttt{ n'.id (F n'))}$$

Inductive consequences with functional inductive types

$$R \ni \texttt{id (lim F)} \longrightarrow \texttt{lim (}\lambda\ \texttt{n. id (F n))}$$
$$R' \setminus R \ni \texttt{id (id x)} \longrightarrow \texttt{id x}$$

$\texttt{id (id (lim F))} \longrightarrow_{R'} \texttt{id (lim F)} \longrightarrow_{R} \texttt{lim (}\lambda\ \texttt{n'.id (F n'))}$

$\downarrow_R$

$\texttt{id (lim (}\lambda\ \texttt{n'. id (F n')))}$

$\downarrow_R$

$\texttt{lim (}\lambda\ \texttt{n''. id ((}\lambda\ \texttt{n'. id (F n')) n''))}$

$\downarrow_\beta$

$\texttt{lim (}\lambda\ \texttt{n''. id (id (F n'')))}$ $\longrightarrow_{R'}$ (to $\texttt{lim (}\lambda\ \texttt{n'.id (F n'))}$)

one needs one step of $\longrightarrow_{R'}$ under λ under `lim`.

Inductive consequences with functional inductive types

```
Symbol n2o : nat → ord

Rules n2o O ⟶ o
      n2o (S x) ⟶ s (n2o x)
```

Inductive consequences with functional inductive types

```
Symbol n2o : nat → ord

Rules n2o O ⟶ o
      n2o (S x) ⟶ s (n2o x)

id (id x) ⟶ id x
```

Inductive consequences with functional inductive types

```
Symbol n2o : nat → ord

Rules n2o O ⟶ o
      n2o (S x) ⟶ s (n2o x)

id (id x) ⟶ id x
```

$l \longrightarrow r$

Inductive consequences with functional inductive types

`Symbol n2o : nat → ord`

```
Rules n2o 0 ⟶ o
      n2o (S x) ⟶ s (n2o x)
```

`id (id x) ⟶ id x` $\quad l \longrightarrow r$

Now, for $\sigma = \{\mathtt{x} \mapsto \mathtt{lim}\ (\lambda\ \mathtt{n.}\ \mathtt{n2o}\ \mathtt{n})\}$ one has:

Inductive consequences with functional inductive types

```
Symbol n2o : nat → ord

Rules n2o 0 ⟶ o
      n2o (S x) ⟶ s (n2o x)
```

`id (id x) ⟶ id x` $\qquad l \longrightarrow r$

Now, for $\sigma = \{$`x` $\mapsto$ `lim (λ n. n2o n)`$\}$ one has:

$l\sigma$`=id (id (lim (λ n. n2o n)))`
`⟶ id (lim (λ n'. id ((λ n. n2o n) n')))`
`⟶ id (lim (λ n'. id (n2o n')))`
`⟶ lim (λ n''. id ((λ n'. id (n2o n')) n''))`
`⟶ lim (λ n''. id (id (n2o n'')))`

$r\sigma$`=id (lim (λ n. n2o n))` $\qquad l\sigma \not\longleftrightarrow^{\star}_{R} r\sigma$
`⟶ lim (λ n'. id ((λ n. n2o n) n'))`
`⟶ lim (λ n'. id (n2o n'))`

Inductive consequences with functional inductive types

`Symbol n2o : nat` $\rightarrow$ `ord`

`Rules n2o O` $\longrightarrow$ `o`
`n2o (S x)` $\longrightarrow$ `s (n2o x)`

`id (id x)` $\longrightarrow$ `id x` $\qquad l \longrightarrow r$

Now, for $\sigma = \{$`x` $\mapsto$ `lim (`λ` n. n2o n)`$\}$ one has:

$l\sigma$=`id (id (lim (`λ` n. n2o n)))`
$\longrightarrow$ `id (lim (`λ` n'. id ((`λ` n. n2o n) n')))`
$\longrightarrow$ `id (lim (`λ` n'. id (n2o n')))`
$\longrightarrow$ `lim (`λ` n''. id ((`λ` n'. id (n2o n')) n''))`
$\longrightarrow$ `lim (`λ` n''. id (id (n2o n'')))`

$r\sigma$=`id (lim (`λ` n. n2o n))` $\qquad l\sigma \not\overset{\star}{\longleftrightarrow}_R r\sigma$
$\longrightarrow$ `lim (`λ` n'. id ((`λ` n. n2o n) n'))`
$\longrightarrow$ `lim (`λ` n'. id (n2o n'))`

Inductive consequences with functional inductive types

Theorem 2

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and

- SN($\longrightarrow_{R'} \cup \longrightarrow_{\beta} \cup \rhd_c$)
- critical pairs are joinable with $\longrightarrow_{R' \setminus R}$ under constructors and $lambda$

then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \sim_{\omega} r\sigma$.

Inductive consequences with functional inductive types

Theorem 2

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and

- SN($\longrightarrow_{R'} \cup \longrightarrow_{\beta} \cup \ \rhd_c$)
- critical pairs are joinable with $\longrightarrow_{R' \setminus R}$ under constructors and $lambda$

then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \sim_{\omega} r\sigma$.

- ω-equivalence: $\sim_{\omega}$

Inductive consequences with functional inductive types

Theorem 2

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and

- SN($\longrightarrow_{R'} \cup \longrightarrow_{\beta} \cup \rhd_c$)
- critical pairs are joinable with $\longrightarrow_{R' \setminus R}$ under constructors and $lambda$

then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \sim_\omega r\sigma$.

- ω-equivalence: $\sim_\omega$ is the least congruence containing $\overset{\star}{\longleftrightarrow}_R$ and ω-rule:

$$\frac{\forall \delta \text{ closed}, \quad s\delta \sim_\omega t\delta}{s \sim_\omega t}$$

Inductive consequences with functional inductive types

Theorem 2

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and

- $\mathsf{SN}(\longrightarrow_{R'} \cup \longrightarrow_{\beta} \cup \rhd_c)$
- critical pairs are joinable with $\longrightarrow_{R' \setminus R}$ under constructors and $lambda$

then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \sim_\omega r\sigma$.

- ω-equivalence: $\sim_\omega$ is the least congruence containing $\stackrel{\star}{\longleftrightarrow}_R$ and ω-rule:

$$\frac{\forall \delta \text{ closed}, \quad s\delta \sim_\omega t\delta}{s \sim_\omega t}$$

- proof by induction on $\longrightarrow_{R'} \cup \longrightarrow_{\beta} \cup \rhd_c$

$\mathtt{S}\ \mathtt{x} \rhd_c \mathtt{x}$

$\mathtt{lim}\ \mathtt{F} \rhd_c \mathtt{F}\ t$ (for every typable t)

Inductive consequences with functional inductive types

Theorem 2

If $R \subseteq R'$ and they are both terminating, confluent and R is complete and

- R and R' are terminating by HORPO or General Schema
- critical pairs are joinable with $\longrightarrow_{R' \setminus R}$ under constructors and $lambda$

then for all $l \longrightarrow r \in R' \setminus R$, for all σ closed, $l\sigma \sim_\omega r\sigma$.

- ω-equivalence: $\sim_\omega$ is the least congruence containing $\overset{\star}{\longleftrightarrow}_R$ and ω-rule:

$$\frac{\forall \delta \text{ closed}, \quad s\delta \sim_\omega t\delta}{s \sim_\omega t}$$

- proof by induction on $\longrightarrow_{R'} \cup \longrightarrow_\beta \cup \rhd_c$

$\texttt{S x} \rhd_c \texttt{x}$
$\texttt{lim F} \rhd_c \texttt{F}\ t$ (for every typable t)

Conclusions

We work on adding rewriting to the Calculus of Constructions.

Conclusions

We work on adding rewriting to the Calculus of Constructions.

- Definitions by rewriting are made from a complete subset and "other" rules.
- We show that (under reasonable conditions) the "other" rules are inductive consequences of the complete subset.

Conclusions

We work on adding rewriting to the Calculus of Constructions.

- Definitions by rewriting are made from a complete subset and "other" rules.
- We show that (under reasonable conditions) the "other" rules are inductive consequences of the complete subset.

General question:
What is the logical power of CC+Rew ?

Conclusions

We work on adding rewriting to the Calculus of Constructions.

- Definitions by rewriting are made from a complete subset and “other” rules.
- We show that (under reasonable conditions) the “other” rules are inductive consequences of the complete subset.

General question:
What is the logical power of CC+Rew ?

Ideally: conservativity of CC+Rew over CIC+K+ext+?

TYPES 2010

Warsaw, October 13-16, 2010

http://types10.mimuw.edu.pl