Inconsistent Type Systems

Alexandre Miquel — PPS & U. Paris 7

Alexandre.Miquel@pps.jussieu.fr

Types Summer School 2005

August 15–26 — Göteborg

Introduction

- System F [Girard 1971]
- 'A theory of types' (Type:Type) [Martin-Löf 1971]
- Inconsistency of system U [Girard 1971]
  Inconsistency of Type:Types comes as a consequence
- Inconsistency of System U^- [Coquand 1991]
- Simplification of Girard's paradox (system U^-) [Hurkens 1995]
- Russell's paradox in systems U/U^- [Miquel 2000]

System $\lambda*$ (Type:Type) [Martin-Löf 71]

System $\lambda*$ (Type:Type) [Martin-Löf 71]

Terms	M, N, T, U	$::=$	$x \mid \lambda x : T \,.\, M \mid M N \mid \mathsf{Type} \mid \Pi x : T \,.\, U$
Contexts	Γ, Δ	$::=$	$[] \mid \Gamma,\ x : T$

System $\lambda*$ (Type:Type) [Martin-Löf 71]

$$\textbf{Terms} \quad M, N, T, U \quad ::= \quad x \quad | \quad \lambda x : T \,.\, M \quad | \quad MN \quad | \quad \text{Type} \quad | \quad \Pi x : T \,.\, U$$

$$\textbf{Contexts} \quad \Gamma, \Delta \quad ::= \quad [] \quad | \quad \Gamma,\ x : T$$

$$\frac{}{\vdash [] \ \text{ctx}} \qquad \frac{\Gamma \vdash T : \text{Type}}{\vdash \Gamma,\ x : T \ \text{ctx}} \ x \notin \mathbf{Dom}(\Gamma)$$

$$\frac{\vdash \Gamma \ \text{ctx}}{\Gamma \vdash x : T} \ (x{:}T) \in \Gamma \qquad \frac{\vdash \Gamma \ \text{ctx}}{\Gamma \vdash \text{Type} : \text{Type}} \qquad \frac{\Gamma,\ x : T \vdash U : \text{Type}}{\Gamma \vdash \Pi x : T \,.\, U : \text{Type}}$$

$$\frac{\Gamma,\ x : T \vdash M : U}{\Gamma \vdash \lambda x : T \,.\, M : \Pi x : T \,.\, U} \qquad \frac{\Gamma \vdash M : \Pi x : T \,.\, U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U\{x := N\}}$$

$$\frac{\Gamma \vdash M : T \qquad \Gamma \vdash T' : \text{Type}}{\Gamma \vdash M : T'} \ T' \approx T$$

System $\lambda*$ (Type:Type) [Martin-Löf 71]

$$\textbf{Terms} \qquad M, N, T, U \quad ::= \quad x \quad | \quad \lambda x : T \,.\, M \quad | \quad MN \quad | \quad \text{Type} \quad | \quad \Pi x : T \,.\, U$$

$$\textbf{Contexts} \qquad \Gamma, \Delta \quad ::= \quad [] \quad | \quad \Gamma,\ x : T$$

$$\frac{}{\vdash [] \ \text{ctx}} \qquad \frac{\Gamma \vdash T : \text{Type}}{\vdash \Gamma,\ x : T \ \text{ctx}}\ {\scriptstyle x \notin \mathbf{Dom}(\Gamma)}$$

$$\frac{\vdash \Gamma \ \text{ctx}}{\Gamma \vdash x : T}\ {\scriptstyle (x:T) \in \Gamma} \qquad \frac{\vdash \Gamma \ \text{ctx}}{\Gamma \vdash \text{Type} : \text{Type}} \qquad \frac{\Gamma,\ x : T \vdash U : \text{Type}}{\Gamma \vdash \Pi x : T \,.\, U : \text{Type}}$$

$$\frac{\Gamma,\ x : T \vdash M : U}{\Gamma \vdash \lambda x : T \,.\, M : \Pi x : T \,.\, U} \qquad \frac{\Gamma \vdash M : \Pi x : T \,.\, U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U\{x := N\}}$$

$$\frac{\Gamma \vdash M : T \qquad \Gamma \vdash T' : \text{Type}}{\Gamma \vdash M : T'}\ {\scriptstyle T' \approx T}$$

- Computationally correct: Church-Rosser, subject reduction

System $\lambda*$ (Type:Type) [Martin-Löf 71]

Terms	M, N, T, U	$::=$	$x \mid \lambda x : T \,.\, M \mid MN \mid \mathsf{Type} \mid \Pi x : T \,.\, U$
Contexts	Γ, Δ	$::=$	$[] \mid \Gamma,\ x : T$

$$\frac{}{\vdash [] \ \mathsf{ctx}} \qquad \frac{\Gamma \vdash T : \mathsf{Type}}{\vdash \Gamma,\ x : T \ \mathsf{ctx}}\ x \notin \mathbf{Dom}(\Gamma)$$

$$\frac{\vdash \Gamma \ \mathsf{ctx}}{\Gamma \vdash x : T}\ (x{:}T) \in \Gamma \qquad \frac{\vdash \Gamma \ \mathsf{ctx}}{\Gamma \vdash \mathsf{Type} : \mathsf{Type}} \qquad \frac{\Gamma,\ x : T \vdash U : \mathsf{Type}}{\Gamma \vdash \Pi x : T \,.\, U : \mathsf{Type}}$$

$$\frac{\Gamma,\ x : T \vdash M : U}{\Gamma \vdash \lambda x : T \,.\, M : \Pi x : T \,.\, U} \qquad \frac{\Gamma \vdash M : \Pi x : T \,.\, U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U\{x := N\}}$$

$$\frac{\Gamma \vdash M : T \qquad \Gamma \vdash T' : \mathsf{Type}}{\Gamma \vdash M : T'}\ T' \approx T$$

- Computationally correct: Church-Rosser, subject reduction
- Logically inconsistent: closed term of type $\bot \equiv \Pi X : \mathsf{Type} \,.\, X$

System $\lambda*$ (Type:Type) [Martin-Löf 71]

Terms	M, N, T, U	$::=$	$x \mid \lambda x : T\,.\,M \mid MN \mid \text{Type} \mid \Pi x : T\,.\,U$
Contexts	Γ, Δ	$::=$	$[] \mid \Gamma,\ x : T$

$$\frac{}{\vdash [] \text{ ctx}} \qquad \frac{\Gamma \vdash T : \text{Type}}{\vdash \Gamma,\ x : T \text{ ctx}}\ x \notin \mathbf{Dom}(\Gamma)$$

$$\frac{\vdash \Gamma \text{ ctx}}{\Gamma \vdash x : T}\ (x{:}T) \in \Gamma \qquad \frac{\vdash \Gamma \text{ ctx}}{\Gamma \vdash \text{Type} : \text{Type}} \qquad \frac{\Gamma,\ x : T \vdash U : \text{Type}}{\Gamma \vdash \Pi x : T\,.\,U : \text{Type}}$$

$$\frac{\Gamma,\ x : T \vdash M : U}{\Gamma \vdash \lambda x : T\,.\,M : \Pi x : T\,.\,U} \qquad \frac{\Gamma \vdash M : \Pi x : T\,.\,U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U\{x := N\}}$$

$$\frac{\Gamma \vdash M : T \qquad \Gamma \vdash T' : \text{Type}}{\Gamma \vdash M : T'}\ T' \approx T$$

- Computationally correct: Church-Rosser, subject reduction
- Logically inconsistent: closed term of type $\bot \equiv \Pi X : \text{Type}\,.\,X$
- Non (weakly) normalising, since:

Fact: Closed terms of type $\bot \equiv \Pi X : \text{Type}\,.\,X$ have no head normal form

Is a type of all types like a set of all sets?

Is a type of all types like a set of all sets?

The analogy between Type:Type and the set of all sets of Cantor-Frege's (inconsistent) set theory is erroneous, since:

Is a type of all types like a set of all sets?

The analogy between Type:Type and the set of all sets of Cantor-Frege's (inconsistent) set theory is erroneous, since:

1. Typing relation and membership relation have not the same status
   - Typing belongs to the meta-language
     $\Rightarrow$ Precondition for an expression to be well-formed
   - Membership is a relation of the language
     $\Rightarrow$ Can be used to form propositions (may be negated)

Is a type of all types like a set of all sets?

The analogy between Type:Type and the set of all sets of Cantor-Frege's (inconsistent) set theory is erroneous, since:

1. Typing relation and membership relation have not the same status
   - Typing belongs to the meta-language
     $\Rightarrow$ Precondition for an expression to be well-formed
   - Membership is a relation of the language
     $\Rightarrow$ Can be used to form propositions (may be negated)
2. No comprehension scheme in Type:Type
   $\Rightarrow$ Cannot form a type of the form $\{x : T \mid P(x)\}$

Is a type of all types like a set of all sets?

The analogy between Type:Type and the set of all sets of Cantor-Frege's (inconsistent) set theory is erroneous, since:

1. Typing relation and membership relation have not the same status
   - Typing belongs to the meta-language
     $\Rightarrow$ Precondition for an expression to be well-formed
   - Membership is a relation of the language
     $\Rightarrow$ Can be used to form propositions (may be negated)
2. No comprehension scheme in Type:Type
   $\Rightarrow$ Cannot form a type of the form $\{x : T \mid P(x)\}$

Historically, the inconsistency of Type:Type has been derived [Girard] from the inconsistency of system U

Is a type of all types like a set of all sets?

The analogy between Type:Type and the set of all sets of Cantor-Frege's (inconsistent) set theory is erroneous, since:

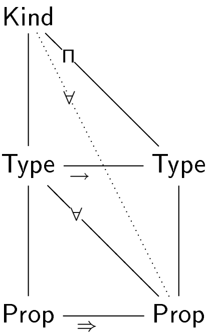
1. Typing relation and membership relation have not the same status
   - Typing belongs to the meta-language
     $\Rightarrow$ Precondition for an expression to be well-formed
   - Membership is a relation of the language
     $\Rightarrow$ Can be used to form propositions (may be negated)
2. No comprehension scheme in Type:Type
   $\Rightarrow$ Cannot form a type of the form $\{x : T \mid P(x)\}$

Historically, the inconsistency of Type:Type has been derived [Girard] from the inconsistency of system U

No cycle in the sorts (Prop : Type : Kind)...

... but two levels of impredicativity (Prop and Type)

Systems U and U^-



$U^- =$ copy of F glued on top of $F\omega$

$U =$ system $U^- +$ (Kind, Prop)-quantification

- Kind = sort for kinds
- Type = sort for constructors
- Prop = sort for proof-terms

Both Type and Prop are impredicative

Higher-level is isomorphic to F:
Type inference/checking is decidable

$$\begin{array}{rcl} \mathcal{S} & = & \{\text{Prop}, \text{Type}, \text{Kind}\} \\ \mathcal{A} & = & \{(\text{Prop} : \text{Type}), (\text{Type} : \text{Kind})\} \\ \mathcal{R} & = & \{\underbrace{(\text{Prop} : \text{Prop}), (\text{Type} : \text{Prop}), (\text{Type}, \text{Type}), (\text{Kind}, \text{Type})}_{\text{system } U^-}, \underbrace{(\text{Kind}, \text{Prop})}_{U \text{ only}}\} \end{array}$$

From system $F\omega$

$$\mathcal{S} = \text{Prop}, \quad \text{Type}$$
$$\mathcal{A} = \text{Prop} : \text{Type}$$
$$\mathcal{R} = (\text{Prop}, \text{Prop}), \quad (\text{Type}, \text{Prop}), \quad (\text{Type}, \text{Type})$$

Kinds	τ, σ	$::=$	Prop	
		$\mid$	$\tau \rightarrow \sigma$	(Type, Type)
Constructors	M, N	$::=$	ξ	
		$\mid$	$\lambda x : \tau . M \quad \mid \quad MN$	(Type, Type)
		$\mid$	$M \Rightarrow N$	(Prop, Prop)
		$\mid$	$\forall x : \tau . M$	(Type, Prop)
Proof-terms	t, u	$::=$	ξ	
		$\mid$	$\lambda \xi : M . t \quad \mid \quad tu$	(Prop, Prop)
		$\mid$	$\lambda x : \tau . t \quad \mid \quad tM$	(Type, Prop)

From system $F\omega$...

$$\mathcal{S} = \text{Prop}, \quad \text{Type}, \quad \text{Kind}$$
$$\mathcal{A} = \text{Prop} : \text{Type}, \quad \text{Type} : \text{Kind}$$
$$\mathcal{R} = (\text{Prop}, \text{Prop}), \quad (\text{Type}, \text{Prop}), \quad (\text{Type}, \text{Type})$$

Kinds	τ, σ	$::=$	$\text{Prop} \mid \alpha$	
		$\mid$	$\tau \to \sigma$	(Type, Type)
Constructors	M, N	$::=$	ξ	
		$\mid$	$\lambda x : \tau \,.\, M \mid MN$	(Type, Type)
		$\mid$	$M \Rightarrow N$	(Prop, Prop)
		$\mid$	$\forall x : \tau \,.\, M$	(Type, Prop)
Proof-terms	t, u	$::=$	ξ	
		$\mid$	$\lambda \xi : M \,.\, t \mid tu$	(Prop, Prop)
		$\mid$	$\lambda x : \tau \,.\, t \mid tM$	(Type, Prop)

From system $F\omega$... to system U^-

$$\mathcal{S} = \text{Prop}, \quad \text{Type}, \quad \text{Kind}$$

$$\mathcal{A} = \text{Prop} : \text{Type}, \quad \text{Type} : \text{Kind}$$

$$\mathcal{R} = (\text{Prop}, \text{Prop}), \quad (\text{Type}, \text{Prop}), \quad (\text{Type}, \text{Type}), \quad (\text{Kind}, \text{Type})$$

Kinds $\tau, \sigma ::= \text{Prop} \mid \alpha$

$\mid \tau \to \sigma$ (Type, Type)

$\mid \Pi\alpha : \text{Type} \,.\, \tau$ (Kind, Type)

Constructors $M, N ::= \xi$

$\mid \lambda x : \tau \,.\, M \mid MN$ (Type, Type)

$\mid \Lambda\alpha \,.\, M \mid M\tau$ (Kind, Type)

$\mid M \Rightarrow N$ (Prop, Prop)

$\mid \forall x : \tau \,.\, M$ (Type, Prop)

Proof-terms $t, u ::= \xi$

$\mid \lambda\xi : M \,.\, t \mid tu$ (Prop, Prop)

$\mid \lambda x : \tau \,.\, t \mid tM$ (Type, Prop)

From system $F\omega$... to system U

$$\begin{array}{lllllll}
\mathcal{S} & = & \text{Prop}, & \text{Type}, & \text{Kind} & & \\
\mathcal{A} & = & \text{Prop} : \text{Type}, & \text{Type} : \text{Kind} & & & \\
\mathcal{R} & = & (\text{Prop}, \text{Prop}), & (\text{Type}, \text{Prop}), & (\text{Type}, \text{Type}), & (\text{Kind}, \text{Type}), & (\text{Kind}, \text{Prop})
\end{array}$$

$$\begin{array}{llllll}
\textbf{Kinds} & \tau, \sigma & ::= & \text{Prop} \quad | \quad \alpha & & \\
& & | & \tau \to \sigma & & (\text{Type}, \text{Type}) \\
& & | & \Pi\alpha : \text{Type} . \tau & & (\text{Kind}, \text{Type}) \\
\textbf{Constructors} & M, N & ::= & \xi & & \\
& & | & \lambda x : \tau . M \quad | \quad MN & & (\text{Type}, \text{Type}) \\
& & | & \Lambda\alpha . M \quad | \quad M\tau & & (\text{Kind}, \text{Type}) \\
& & | & M \Rightarrow N & & (\text{Prop}, \text{Prop}) \\
& & | & \forall x : \tau . M & & (\text{Type}, \text{Prop}) \\
& & | & \forall\alpha : \text{Type} . M & & (\text{Kind}, \text{Prop}) \\
\textbf{Proof-terms} & t, u & ::= & \xi & & \\
& & | & \lambda\xi : M . t \quad | \quad tu & & (\text{Prop}, \text{Prop}) \\
& & | & \lambda x : \tau . t \quad | \quad tM & & (\text{Type}, \text{Prop}) \\
& & | & \lambda\alpha : \text{Type} . t \quad | \quad t\tau & & (\text{Kind}, \text{Prop})
\end{array}$$

Examples

(Kind, Type)	$\Pi\alpha : \text{Type} \ldots$	Polymorphism in data types
(Type : Prop)	$\forall x : \tau \ldots$	Quantification over all objects (of a given type)
(Kind, Prop)	$\forall\alpha : \text{Type} \ldots$	Quantification over all types

Examples

(Kind, Type)	$\Pi\alpha : \mathsf{Type} \ldots$	Polymorphism in data types
(Type : Prop)	$\forall x : \tau \ldots$	Quantification over all objects (of a given type)
(Kind, Prop)	$\forall\alpha : \mathsf{Type} \ldots$	Quantification over all types

$$\mathsf{Nat} \quad := \quad \Pi\alpha : \mathsf{Type} \,.\, (\alpha \to (\alpha \to \alpha) \to \alpha) \quad : \quad \mathsf{Type}$$

Examples

(Kind, Type)	$\Pi\alpha : \text{Type} \ldots$	Polymorphism in data types
(Type : Prop)	$\forall x : \tau \ldots$	Quantification over all objects (of a given type)
(Kind, Prop)	$\forall\alpha : \text{Type} \ldots$	Quantification over all types

$$\text{Nat} \quad := \quad \Pi\alpha : \text{Type} \,.\, (\alpha \to (\alpha\to\alpha) \to \alpha) \quad : \quad \text{Type}$$

$$\text{id} \quad := \quad \lambda\alpha : \text{Type} \,.\, \lambda x : \alpha \,.\, x \quad : \quad \Pi\alpha : \text{Type} \,.\, (\alpha \to \alpha)$$

Examples

(Kind, Type)	$\Pi\alpha : \mathsf{Type}\ldots$	Polymorphism in data types
(Type : Prop)	$\forall x : \tau \ldots$	Quantification over all objects (of a given type)
(Kind, Prop)	$\forall\alpha : \mathsf{Type}\ldots$	Quantification over all types

$$\mathsf{Nat} \quad := \quad \Pi\alpha : \mathsf{Type}\,.\,(\alpha \to (\alpha{\to}\alpha) \to \alpha) \quad : \quad \mathsf{Type}$$

$$\mathsf{id} \quad := \quad \lambda\alpha : \mathsf{Type}\,.\,\lambda x : \alpha\,.\,x \quad : \quad \Pi\alpha : \mathsf{Type}\,.\,(\alpha \to \alpha)$$

$$x \;=_\alpha\; y \quad := \quad \forall p : (\alpha{\to}\mathsf{Prop})\,.\;(p\,x \Rightarrow p\,y) \quad : \quad \mathsf{Prop}$$

Examples

(Kind, Type)	$\Pi\alpha : \mathsf{Type} \ldots$	Polymorphism in data types
(Type : Prop)	$\forall x : \tau \ldots$	Quantification over all objects (of a given type)
(Kind, Prop)	$\forall\alpha : \mathsf{Type} \ldots$	Quantification over all types

$$\mathsf{Nat} \quad := \quad \Pi\alpha : \mathsf{Type} \,.\, (\alpha \to (\alpha \to \alpha) \to \alpha) \quad : \quad \mathsf{Type}$$

$$\mathsf{id} \quad := \quad \lambda\alpha : \mathsf{Type} \,.\, \lambda x : \alpha \,.\, x \quad : \quad \Pi\alpha : \mathsf{Type} \,.\, (\alpha \to \alpha)$$

$$x \ =_{\alpha} \ y \quad := \quad \forall p : (\alpha \to \mathsf{Prop}) \,.\ (p\,x \Rightarrow p\,y) \quad : \quad \mathsf{Prop}$$

$$\forall\alpha : \mathsf{Type} \,. \quad \forall x : \alpha \,. \quad \mathsf{id}\ \alpha\ x \ =_{\alpha}\ x \quad : \quad \mathsf{Prop}$$

Examples

(Kind, Type)	$\Pi\alpha : \mathsf{Type} \ldots$	Polymorphism in data types
(Type : Prop)	$\forall x : \tau \ldots$	Quantification over all objects (of a given type)
(Kind, Prop)	$\forall \alpha : \mathsf{Type} \ldots$	Quantification over all types

$$\mathsf{Nat} \quad := \quad \Pi\alpha : \mathsf{Type} . (\alpha \to (\alpha{\to}\alpha) \to \alpha) \quad : \quad \mathsf{Type}$$

$$\mathsf{id} \quad := \quad \lambda\alpha : \mathsf{Type} . \lambda x : \alpha . x \quad : \quad \Pi\alpha : \mathsf{Type} . (\alpha \to \alpha)$$

$$x \; =_\alpha \; y \quad := \quad \forall p : (\alpha{\to}\mathsf{Prop}) . \; (p\, x \Rightarrow p\, y) \quad : \quad \mathsf{Prop}$$

$$\forall\alpha : \mathsf{Type} . \quad \forall x : \alpha . \quad \mathsf{id}\ \alpha\ x \; =_\alpha \; x \quad : \quad \mathsf{Prop}$$

$$\lambda\alpha : \mathsf{Type} . \quad \lambda x : \alpha . \quad \lambda p : (\alpha{\to}\mathsf{Prop}) . \quad \lambda\xi : p\, x . \xi \quad : \quad \ldots$$

Hurkens' paradox in system U^-

Hurkens' paradox in system U^-

For any kind τ : Type write: $\mathfrak{P}(\tau) \;:=\; \tau \to \mathrm{Prop}$

$\bot$	: Prop	:=	$\forall a : \mathrm{Prop} \,.\, a$
$\neg$	: Prop $\to$ Prop	:=	$\lambda a : \mathrm{Prop} \,.\, a \Rightarrow \bot$
$\mathbb{U}$	: Type	:=	$\Pi\alpha : \mathrm{Type} \,.\, \Big((\mathfrak{P}(\mathfrak{P}(\alpha)) \to \alpha) \to \mathfrak{P}(\mathfrak{P}(\alpha)) \Big)$
i	: $\mathfrak{P}(\mathfrak{P}(\mathbb{U})) \to \mathbb{U}$	:=	$\lambda q : \mathfrak{P}(\mathfrak{P}(\mathbb{U})) \,.\, \lambda\alpha : \mathrm{Type} \,.\, \lambda f : (\mathfrak{P}(\mathfrak{P}(\alpha)) \to \alpha) \,.$ $\lambda p : \mathfrak{P}(\alpha) \,.\; q \; (\lambda x : \mathbb{U} \,.\, p \; (f \; (x \; \alpha \; f)))$
j	: $\mathbb{U} \to \mathfrak{P}(\mathfrak{P}(\mathbb{U}))$	:=	$\lambda x : \mathbb{U} \,.\; x \; \mathbb{U} \; i$
Q	: $\mathfrak{P}(\mathfrak{P}(\mathbb{U}))$	:=	$\lambda p : \mathfrak{P}(\mathbb{U}) \,.\; \forall x : \mathbb{U} \,.\, (j \; x \; p \Rightarrow p \; x)$
C	: $\mathfrak{P}(\mathbb{U})$	:=	$\lambda y : \mathbb{U} \,.\; \neg\forall p : \mathfrak{P}(\mathbb{U}) \,.\, (j \; y \; p \Rightarrow p \; (i \; (j \; y)))$
B	: $\mathbb{U}$	:=	$i \; Q$
lem_1	: $Q \; C$	:=	$\lambda x : U \,.\, \lambda\xi^{jxC} \,.\, \lambda\zeta^{\forall p : \mathfrak{P}(\mathbb{U}) \,.\, (jxp \Rightarrow p(i(jx)))} \,.$ $\zeta \; C \; \xi \; (\lambda p : \mathfrak{P}(\mathbb{U}) \,.\, \zeta \; (\lambda y : \mathbb{U} \,.\, p \; (i \; (j \; y))))$
A	: Prop	:=	$\forall p : \mathfrak{P}(\mathbb{U}) \,.\; (Q \; p \Rightarrow p \; B)$
lem_2	: $\neg A$	:=	$\lambda\xi^{A} \,.\, \xi \; C \; \mathrm{lem}_1 \; (\lambda p : \mathfrak{P}(\mathbb{U}) \,.\, \xi \; (\lambda y : \mathbb{U} \,.\, p \; (i \; (j \; y))))$
lem_3	: A	:=	$\lambda p : \mathfrak{P}(\mathbb{U}) \,.\, \lambda\xi^{Qp} \,.\, \xi \; B \; (\lambda x : \mathbb{U} \,.\, \xi \; (i \; (j \; x)))$
paradox	: $\bot$	:=	$\mathrm{lem}_2 \; \mathrm{lem}_3$

Encoding sets as pointed graphs

Encoding sets as pointed graphs

Pointed graph $=$ triple (X, A, a) where

- X : Type — the type of vertices
- $A : X \to X \to \mathrm{Prop}$ — the (local) membership relation
- $a : X$ — the root

Encoding sets as pointed graphs

Pointed graph $=$ triple (X, A, a) where

- X : Type — the type of vertices
- $A : X \to X \to \text{Prop}$ — the (local) membership relation
- $a : X$ — the root

$A(x, y)$ is represented by $\bullet_x \leftarrow \bullet_y$, and the root a by $\bullet_a$

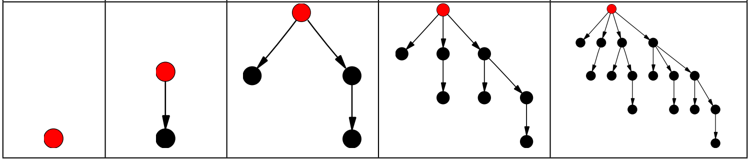
Encoding sets as pointed graphs

Pointed graph $=$ triple (X, A, a) where

- X : Type — the type of vertices
- $A : X \to X \to \text{Prop}$ — the (local) membership relation
- $a : X$ — the root

$A(x, y)$ is represented by $\bullet_x \leftarrow \bullet_y$, and the root a by $\bullet_a$

$0 = \varnothing$	$1 = \{0\}$	$2 = \{0; 1\}$	$3 = \{0; 1; 2\}$	$4 = \{0; 1; 2; 3\}$



Identifying related pointed graphs

A set can be represented by several non-isomorphic pointed graphs

Identifying related pointed graphs

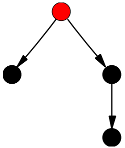
A set can be represented by several non-isomorphic pointed graphs

Example: the set $2 = \{\varnothing; \{\varnothing\}\}$

Identifying related pointed graphs

A set can be represented by several non-isomorphic pointed graphs

Example: the set $2 = \{\varnothing; \{\varnothing\}\}$

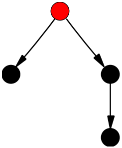


no sharing (tree)

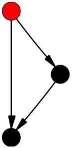
Identifying related pointed graphs

A set can be represented by several non-isomorphic pointed graphs

Example: the set $2 = \{\varnothing; \{\varnothing\}\}$



no sharing (tree)

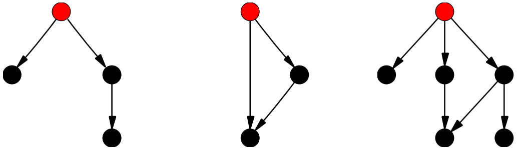


with sharing

Identifying related pointed graphs

A set can be represented by several non-isomorphic pointed graphs

Example: the set $2 = \{\emptyset; \{\emptyset\}\}$

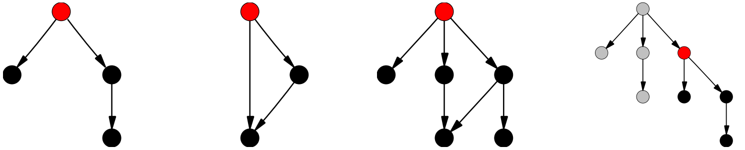


no sharing (tree) with sharing duplicate elements

Identifying related pointed graphs

A set can be represented by several non-isomorphic pointed graphs

Example: the set $2 = \{\varnothing; \{\varnothing\}\}$

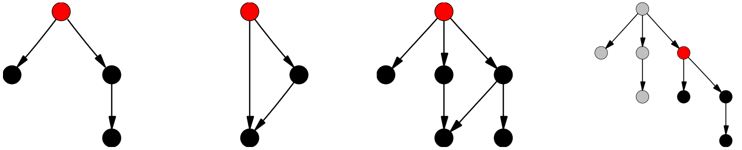


no sharing (tree) with sharing duplicate elements unreachable parts

Identifying related pointed graphs

A set can be represented by several non-isomorphic pointed graphs

Example: the set $2 = \{\varnothing; \{\varnothing\}\}$



no sharing (tree) | with sharing | duplicate elements | unreachable parts

+ Problems related to (possible) non well-foundedness

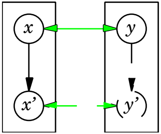
Extensional equality as bisimilarity

Extensional equality as bisimilarity

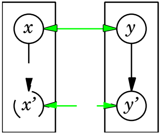
$R : X \to Y \to \text{Prop}$ bisimulation between (X, A, a) and (Y, B, b) if:

1. $\forall x, x' : X \quad \forall y : Y \quad \big(A(x', x) \wedge R(x, y) \;\Rightarrow\; \exists y' : Y \; (R(x', y') \wedge B(y', y))\big)$
2. $\forall x : X \quad \forall y, y' : Y \quad \big(B(y', y) \wedge R(x, y) \;\Rightarrow\; \exists x' : X \; (R(x', y') \wedge A(x', x))\big)$
3. $R(a, b)$

(1)



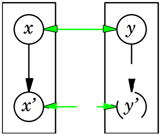
(2)

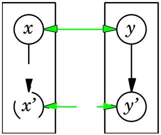


Extensional equality as bisimilarity

$R : X \to Y \to \mathrm{Prop}$ bisimulation between (X, A, a) and (Y, B, b) if:

1. $\forall x, x' {:} X \quad \forall y {:} Y \quad \big(A(x', x) \wedge R(x, y) \;\Rightarrow\; \exists y' {:} Y \;\; (R(x', y') \wedge B(y', y))\big)$
2. $\forall x {:} X \quad \forall y, y' {:} Y \quad \big(B(y', y) \wedge R(x, y) \;\Rightarrow\; \exists x' {:} X \;\; (R(x', y') \wedge A(x', x))\big)$
3. $R(a, b)$

(1) 

(2) 

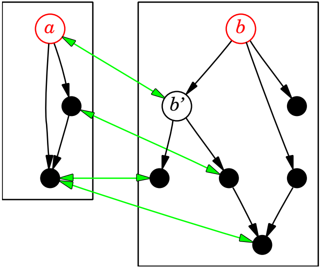
$(X, A, a) \approx (Y, B, b) \quad \equiv \quad \exists R : X \to Y \to \mathrm{Prop} \quad \text{bisimulation}$

Membership as shifted bisimilarity

$$(X, A, a) \in (Y, B, b) \quad \equiv \quad \exists b' : Y \ \ ((X, A, a) \approx (Y, B, b') \ \wedge \ B(b', b))$$

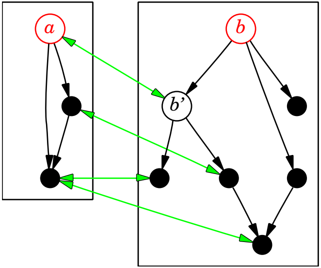
Membership as shifted bisimilarity

$$(X, A, a) \in (Y, B, b) \quad \equiv \quad \exists b' : Y \ \ ((X, A, a) \approx (Y, B, b') \ \wedge \ B(b', b))$$



Membership as shifted bisimilarity

$$(X, A, a) \in (Y, B, b) \quad \equiv \quad \exists b' : Y \quad ((X, A, a) \approx (Y, B, b') \;\wedge\; B(b', b))$$



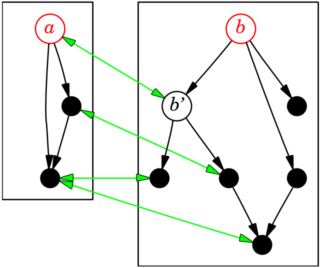
- Compatibility of $\in$ w.r.t $\approx$

  $G_1 \approx G_2 \quad \wedge \quad G_2 \in G_3 \quad \Rightarrow \quad G_1 \in G_3$

  $G_1 \in G_2 \quad \wedge \quad G_2 \approx G_3 \quad \Rightarrow \quad G_1 \in G_3$

Membership as shifted bisimilarity

$$(X, A, a) \in (Y, B, b) \quad \equiv \quad \exists b' : Y \;\; \big((X, A, a) \approx (Y, B, b') \;\wedge\; B(b', b)\big)$$



- Compatibility of $\in$ w.r.t $\approx$

  $G_1 \approx G_2 \quad \wedge \quad G_2 \in G_3 \quad \Rightarrow \quad G_1 \in G_3$

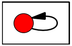
  $G_1 \in G_2 \quad \wedge \quad G_2 \approx G_3 \quad \Rightarrow \quad G_1 \in G_3$

- Extensionality of $\approx$ w.r.t. $\in$

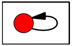
  $\forall G \; (G \in G_1 \;\Leftrightarrow\; G \in G_2) \quad \Rightarrow \quad G_1 \approx G_2$

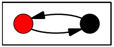
Non well-founded sets

Non well-founded sets

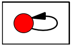
 represents a set x such that $x = \{x\}$

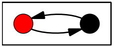
Non well-founded sets

 represents a set x such that $x = \{x\}$

 represents a set y such that $y = \{z\}$ and $z = \{y\}$ for some z

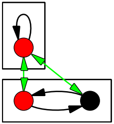
Non well-founded sets

 represents a set x such that $x = \{x\}$

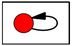
 represents a set y such that $y = \{z\}$ and $z = \{y\}$ for some z

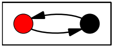
Since there is a bisimulation, we have

$x = y = z = \{x\} = \{y\} = \{z\}$



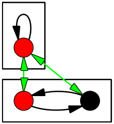
Non well-founded sets

 represents a set x such that $x = \{x\}$

 represents a set y such that $y = \{z\}$ and $z = \{y\}$ for some z

Since there is a bisimulation, we have

$x = y = z = \{x\} = \{y\} = \{z\}$



Sets as pointed graphs $+$ Equality as a bisimulation

$\Rightarrow$ Interprets the Anti-Foundation Axiom (*AFA*) [P. Aczel]

The universal type for representing pointed graphs

The universal type for representing pointed graphs

$$\text{Let} \quad U \quad := \quad \Big(\Pi\, T : \text{Type}\,.\,(T \to T \to \text{Prop}) \to T \to \text{Prop}\Big) \to \text{Prop}$$

$$\begin{aligned} \text{and} \quad i \quad &: \quad \Pi X : \text{Type}\,.\,(X \to X \to \text{Prop}) \to X \to U \\ &:= \quad \lambda X, A, a\,.\,\lambda f\,.\,f\ X\ A\ a \end{aligned}$$

The universal type for representing pointed graphs

$$\text{Let} \quad U \quad := \quad \Big(\underline{\Pi T : \text{Type}} \,.\, (T \to T \to \text{Prop}) \to T \to \text{Prop}\Big) \to \text{Prop}$$

$$\text{and} \quad i \quad : \quad \Pi X : \text{Type} \,.\, (X \to X \to \text{Prop}) \to X \to U$$

$$:= \quad \lambda X, A, a \,.\, \lambda f \,.\, f \;\; X \;\; A \;\; a$$

- Higher-level impredicativity (Kind, Type) ensures that U : Type

The universal type for representing pointed graphs

$$\text{Let} \quad U \quad := \quad \left(\underline{\Pi T : \mathsf{Type}} \,.\, (T \to T \to \mathsf{Prop}) \to T \to \mathsf{Prop}\right) \to \mathsf{Prop}$$

$$\begin{aligned} \text{and} \quad i \quad &: \quad \Pi X : \mathsf{Type} \,.\, (X \to X \to \mathsf{Prop}) \to X \to U \\ &:= \quad \lambda X, A, a \,.\, \lambda f \,.\, f \; X \; A \; a \end{aligned}$$

- Higher-level impredicativity $(\mathsf{Kind}, \mathsf{Type})$ ensures that $U : \mathsf{Type}$
- The map i is an embedding of pointed graphs into U

$$i(X, A, a) = i(Y, B, b) \quad \Rightarrow \quad (X, A, a) \approx (Y, B, b)$$

The universal type for representing pointed graphs

Let $U \quad := \quad \Big(\underline{\Pi T : \mathsf{Type}} \,.\, (T \to T \to \mathsf{Prop}) \to T \to \mathsf{Prop} \Big) \to \mathsf{Prop}$

and $i \quad : \quad \Pi X : \mathsf{Type} \,.\, (X \to X \to \mathsf{Prop}) \to X \to U$
$\quad := \quad \lambda X, A, a \,.\, \lambda f \,.\, f \;\; X \;\; A \;\; a$

- Higher-level impredicativity $(\mathsf{Kind}, \mathsf{Type})$ ensures that $U : \mathsf{Type}$
- The map i is an embedding of pointed graphs into U

$$i(X, A, a) = i(Y, B, b) \quad \Rightarrow \quad (X, A, a) \approx (Y, B, b)$$

- The map i is not surjective:

$$r : U \;=\; \lambda f \,.\, \bot \quad \text{is outside the codomain of } i$$

Translating equivalence and membership on U

$$u \approx v \quad := \quad \exists X, A, a \quad \exists Y, B, b \quad \big(u = i(X, A, a) \ \wedge \ v = i(Y, B, b) \ \wedge \ (X, A, a) \approx (Y, B, b)\big)$$

$$u \in v \quad := \quad \exists X, A, a \quad \exists Y, B, b \quad \big(u = i(X, A, a) \ \wedge \ v = i(Y, B, b) \ \wedge \ (X, A, a) \in (Y, B, b)\big)$$

Translating equivalence and membership on U

$$u \approx v \quad := \quad \exists X, A, a \;\; \exists Y, B, b \;\; \big(u = i(X, A, a) \;\wedge\; v = i(Y, B, b) \;\wedge\; (X, A, a) \approx (Y, B, b)\big)$$

$$u \in v \quad := \quad \exists X, A, a \;\; \exists Y, B, b \;\; \big(u = i(X, A, a) \;\wedge\; v = i(Y, B, b) \;\wedge\; (X, A, a) \in (Y, B, b)\big)$$

$$\mathrm{set}(u) \quad := \quad \exists X, A, a \quad u = i(X, A, a) \qquad (\equiv \text{ codomain of } i)$$

Translating equivalence and membership on U

$$u \approx v \quad := \quad \exists X, A, a \quad \exists Y, B, b \\ \left(u = i(X, A, a) \ \wedge \ v = i(Y, B, b) \ \wedge \ (X, A, a) \approx (Y, B, b)\right)$$

$$u \in v \quad := \quad \exists X, A, a \quad \exists Y, B, b \\ \left(u = i(X, A, a) \ \wedge \ v = i(Y, B, b) \ \wedge \ (X, A, a) \in (Y, B, b)\right)$$

$$\mathrm{set}(u) \quad := \quad \exists X, A, a \qquad u = i(X, A, a) \qquad (\equiv \ \text{codomain of } i)$$

- $\approx$ (on U) is now a partial equivalence relation

Translating equivalence and membership on U

$$u \approx v \quad := \quad \exists X, A, a \quad \exists Y, B, b$$
$$(u = i(X, A, a) \ \wedge \ v = i(Y, B, b) \ \wedge \ (X, A, a) \approx (Y, B, b))$$

$$u \in v \quad := \quad \exists X, A, a \quad \exists Y, B, b$$
$$(u = i(X, A, a) \ \wedge \ v = i(Y, B, b) \ \wedge \ (X, A, a) \in (Y, B, b))$$

$$\mathrm{set}(u) \quad := \quad \exists X, A, a \qquad u = i(X, A, a) \qquad (\equiv \ \text{codomain of } i)$$

- $\approx$ (on U) is now a partial equivalence relation
- Relations $\approx$ and $\in$ are defined on elements $u : U$ s.t. $\mathrm{set}(u)$

Translating equivalence and membership on U

$$u \approx v \quad := \quad \exists X, A, a \;\; \exists Y, B, b \;\; (u = i(X, A, a) \;\wedge\; v = i(Y, B, b) \;\wedge\; (X, A, a) \approx (Y, B, b))$$

$$u \in v \quad := \quad \exists X, A, a \;\; \exists Y, B, b \;\; (u = i(X, A, a) \;\wedge\; v = i(Y, B, b) \;\wedge\; (X, A, a) \in (Y, B, b))$$

$$\mathrm{set}(u) \quad := \quad \exists X, A, a \quad u = i(X, A, a) \qquad (\equiv \text{ codomain of } i)$$

- $\approx$ (on U) is now a partial equivalence relation
- Relations $\approx$ and $\in$ are defined on elements $u : U$ s.t. $\mathrm{set}(u)$
- Other properties of $\approx$ and $\in$ are kept (compatibility, extensionality)

Translating equivalence and membership on U

$$u \approx v \quad := \quad \exists X, A, a \quad \exists Y, B, b \quad (u = i(X, A, a) \ \wedge \ v = i(Y, B, b) \ \wedge \ (X, A, a) \approx (Y, B, b))$$

$$u \in v \quad := \quad \exists X, A, a \quad \exists Y, B, b \quad (u = i(X, A, a) \ \wedge \ v = i(Y, B, b) \ \wedge \ (X, A, a) \in (Y, B, b))$$

$$\mathrm{set}(u) \quad := \quad \exists X, A, a \qquad u = i(X, A, a) \qquad (\equiv \ \text{codomain of } i)$$

- $\approx$ (on U) is now a partial equivalence relation
- Relations $\approx$ and $\in$ are defined on elements $u : U$ s.t. $\mathrm{set}(u)$
- Other properties of $\approx$ and $\in$ are kept (compatibility, extensionality)
- Exists some object $r : U$ such that $\neg\mathrm{set}(r)$

The unbounded comprehension scheme

Let $P : U \to \mathrm{Prop}$ be a predicate over objects of type U

The unbounded comprehension scheme

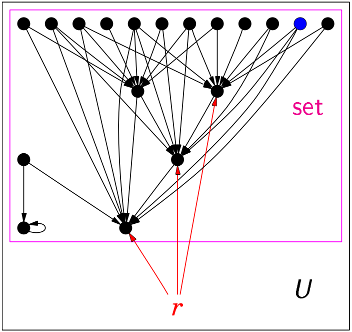
Let $P : U \longrightarrow \mathrm{Prop}$ be a predicate over objects of type U

We assume P extensional: $\forall u, u' : U \,.\, (P(u) \wedge u \approx u' \Rightarrow P(u'))$

The unbounded comprehension scheme

Let $P : U \to \mathrm{Prop}$ be a predicate over objects of type U

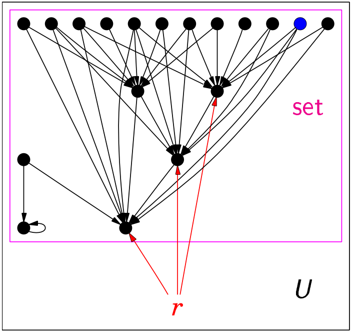
We assume P extensional: $\forall u, u' : U \,.\, (P(u) \wedge u \approx u' \Rightarrow P(u'))$



The unbounded comprehension scheme

Let $P : U \rightarrow \text{Prop}$ be a predicate over objects of type U

We assume P extensional: $\forall u, u' : U \,.\, (P(u) \land u \approx u' \Rightarrow P(u'))$

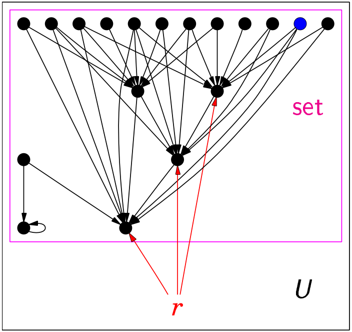


1. Connect r to all $\bullet$ s.t. $P(\bullet)$

The unbounded comprehension scheme

Let $P : U \to \mathrm{Prop}$ be a predicate over objects of type U

We assume P extensional: $\forall u, u' : U \,.\, (P(u) \wedge u \approx u' \Rightarrow P(u'))$

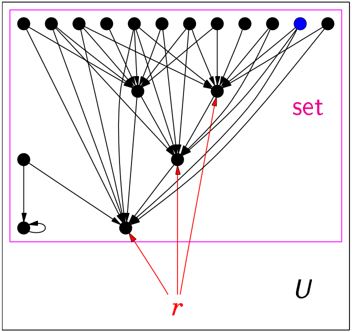


1. Connect r to all • s.t. $P(\bullet)$
2. Let $R_P = \{\to\} \cup \{\to\}$

The unbounded comprehension scheme

Let $P : U \to \text{Prop}$ be a predicate over objects of type U

We assume P extensional: $\forall u, u' : U \,.\, (P(u) \wedge u \approx u' \Rightarrow P(u'))$

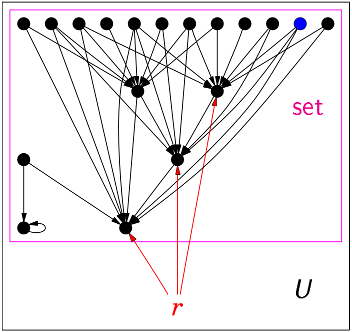


1. Connect r to all $\bullet$ s.t. $P(\bullet)$
2. Let $R_P = \{\to\} \cup \{\to\}$
3. Reflect (U, R_P, r) into U, setting
   $\text{fold}(P) = i(U, R_P, r) \qquad (\equiv \bullet)$

The unbounded comprehension scheme

Let $P : U \to \text{Prop}$ be a predicate over objects of type U

We assume P extensional: $\forall u, u' : U \,.\, (P(u) \wedge u \approx u' \Rightarrow P(u'))$



1. Connect r to all $\bullet$ s.t. $P(\bullet)$
2. Let $R_P = \{\to\} \cup \{\to\}$
3. Reflect (U, R_P, r) into U, setting

$$\text{fold}(P) = i(U, R_P, r) \qquad (\equiv \bullet)$$

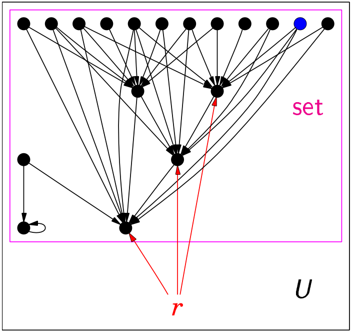
$\Rightarrow$ Relies on the embedding property

$$(X, A, a) \approx (U, \in, i(X, A, a))$$

The unbounded comprehension scheme

Let $P : U \to \mathrm{Prop}$ be a predicate over objects of type U

We assume P extensional: $\forall u, u' : U \,.\, (P(u) \wedge u \approx u' \Rightarrow P(u'))$



1. Connect r to all $\bullet$ s.t. $P(\bullet)$
2. Let $R_P = \{\to\} \cup \{\to\}$
3. Reflect (U, R_P, r) into U, setting

$$\mathrm{fold}(P) = i(U, R_P, r) \qquad (\equiv \bullet)$$

$\Rightarrow$ Relies on the embedding property

$$(X, A, a) \approx (U, \in, i(X, A, a))$$

Fact (Unbounded comprehension)

$\forall u : U \,.\, (u \in i(U, R_P, r) \Leftrightarrow P(u))$ (if P is extensional)

Cantor-Frege's set theory in systems U/U^{-}

Type U $+$ two relations $\approx$ and $\in$

Cantor-Frege's set theory in systems U/U^-

Type U $+$ two relations $\approx$ and $\in$

- $\in$ is compatible w.r.t. $\approx$

Cantor-Frege's set theory in systems U/U^-

Type U $+$ two relations $\approx$ and $\in$

- $\in$ is compatible w.r.t. $\approx$
- $\approx$ is extensional w.r.t. $\in$

Cantor-Frege's set theory in systems U/U^-

Type U + two relations $\approx$ and $\in$

- $\in$ is compatible w.r.t. $\approx$
- $\approx$ is extensional w.r.t. $\in$
- The unbounded comprehension scheme is derivable

Cantor-Frege's set theory in systems U/U^-

Type U $+$ two relations $\approx$ and $\in$

- $\in$ is compatible w.r.t. $\approx$
- $\approx$ is extensional w.r.t. $\in$
- The unbounded comprehension scheme is derivable

$\Rightarrow$ An embedding of Cantor-Frege's (insonsistent) set theory into U/U^-

Cantor-Frege's set theory in systems U/U^-

Type U $+$ two relations $\approx$ and $\in$

- $\in$ is compatible w.r.t. $\approx$
- $\approx$ is extensional w.r.t. $\in$
- The unbounded comprehension scheme is derivable

$\Rightarrow$ An embedding of Cantor-Frege's (insonsistent) set theory into U/U^-
All the usual paradoxes can be derived (Burali-Forti, Russell, ...)

Cantor-Frege's set theory in systems U/U^-

Type U $+$ two relations $\approx$ and $\in$

- $\in$ is compatible w.r.t. $\approx$
- $\approx$ is extensional w.r.t. $\in$
- The unbounded comprehension scheme is derivable

$\Rightarrow$ An embedding of Cantor-Frege's (insonsistent) set theory into U/U^-

All the usual paradoxes can be derived (Burali-Forti, Russell, ...)

Russell's paradox: Consider the set $\mathsf{fold}(\lambda x\,.\,x \notin x)\ldots$

Cantor-Frege's set theory in systems U/U^-

Type U + two relations $\approx$ and $\in$

- $\in$ is compatible w.r.t. $\approx$
- $\approx$ is extensional w.r.t. $\in$
- The unbounded comprehension scheme is derivable

$\Rightarrow$ An embedding of Cantor-Frege's (insonsistent) set theory into U/U^-
All the usual paradoxes can be derived (Burali-Forti, Russell, ...)

Russell's paradox: Consider the set $\text{fold}(\lambda x \,.\, x \notin x)\ldots$

Remark: The formalization has been presented in system U

If we only consider pointed graphs based on $X = U$, we can drop the $(\text{Kind}, \text{Prop})$-quantification, thus restricting to system U^-

Why is system U^- inconsistent?

Why is system U^- inconsistent?

$$
\begin{array}{llcll}
\textbf{Kinds} & \tau, \sigma & ::= & \mathsf{Prop} \quad | \quad \alpha & \\
 & & | & \tau \longrightarrow \sigma & (\mathsf{Type},\ \mathsf{Type}) \\
 & & | & \Pi\alpha : \mathsf{Type} \,.\, \tau & (\mathsf{Kind},\ \mathsf{Type}) \\
\textbf{Constructors} & M, N & ::= & \xi & \\
 & & | & \lambda x : \tau \,.\, M \quad | \quad MN & (\mathsf{Type},\ \mathsf{Type}) \\
 & & | & \Lambda\alpha \,.\, M \quad | \quad M\tau & (\mathsf{Kind},\ \mathsf{Type}) \\
 & & | & M \Rightarrow N & (\mathsf{Prop},\ \mathsf{Prop}) \\
 & & | & \forall x : \tau \,.\, M & (\mathsf{Type},\ \mathsf{Prop}) \\
\textbf{Proof-terms} & t, u & ::= & \xi & \\
 & & | & \lambda x : M \,.\, t \quad | \quad tu & (\mathsf{Prop},\ \mathsf{Prop}) \\
 & & | & \lambda x : \tau \,.\, t \quad | \quad tM & (\mathsf{Type},\ \mathsf{Prop})
\end{array}
$$

Why is system U^- inconsistent?

Kinds	τ, σ	$::=$	$\mathrm{Prop} \mid \alpha$	
		$\mid$	$\tau \rightarrow \sigma$	(Type, Type)
		$\mid$	$\Pi\alpha : \mathrm{Type} \,.\, \tau$	(Kind, Type)
Constructors	M, N	$::=$	ξ	
		$\mid$	$\lambda x : \tau \,.\, M \mid MN$	(Type, Type)
		$\mid$	$\Lambda\alpha \,.\, M \mid M\tau$	(Kind, Type)
		$\mid$	$M \Rightarrow N$	(Prop, Prop)
		$\mid$	$\forall x : \tau \,.\, M$	(Type, Prop)
Proof-terms	t, u	$::=$	ξ	
		$\mid$	$\lambda x : M \,.\, t \mid tu$	(Prop, Prop)
		$\mid$	$\lambda x : \tau \,.\, t \mid tM$	(Type, Prop)

Why is system U^- inconsistent?

Kinds	τ, σ	$::=$	$\mathsf{Prop} \mid \alpha$	
		$\mid$	$\tau \longrightarrow \sigma$	(Type, Type)
		$\mid$	$\Pi\alpha : \mathsf{Type} \,.\, \tau$	(Kind, Type)
Constructors	M, N	$::=$	ξ	
		$\mid$	$\lambda x : \tau \,.\, M \mid MN$	(Type, Type)
		$\mid$	$\Lambda\alpha \,.\, M \mid M\tau$	(Kind, Type)
		$\mid$	$M \Rightarrow N$	(Prop, Prop)
		$\mid$	$\forall x : \tau \,.\, M$	(Type, Prop)
Proof-terms	t, u	$::=$	ξ	
		$\mid$	$\lambda x \,.\, t \mid tu$	(Prop, Prop)

- (Type, Prop)-abstraction/application can be erased

Why is system U^- inconsistent?

Kinds	τ, σ	$::=$	$\mathsf{Prop} \mid \alpha$	
		$\mid$	$\tau \longrightarrow \sigma$	(Type, Type)
		$\mid$	$\Pi\alpha : \mathsf{Type} \,.\, \tau$	(Kind, Type)
Constructors	M, N	$::=$	ξ	
		$\mid$	$\lambda x : \tau \,.\, M \mid MN$	(Type, Type)
		$\mid$	$\Lambda\alpha \,.\, M \mid M\tau$	(Kind, Type)
		$\mid$	$M \Rightarrow N$	(Prop, Prop)
		$\mid$	$\forall x : \tau \,.\, M$	(Type, Prop)
Proof-terms	t, u	$::=$	ξ	
		$\mid$	$\lambda x \,.\, t \mid tu$	(Prop, Prop)

- (Type, Prop)-abstraction/application can be erased

Why is system U^- inconsistent?

Kinds	τ, σ	$::=$	$\mathsf{Prop} \mid \alpha$	
		$\mid$	$\tau \to \sigma$	(Type, Type)
		$\mid$	$\Pi\alpha : \mathsf{Type} . \tau$	(Kind, Type)
Constructors	M, N	$::=$	ξ	
		$\mid$	$\lambda x . M \mid MN$	(Type, Type)
		$\mid$		(Kind, Type)
		$\mid$	$M \Rightarrow N$	(Prop, Prop)
		$\mid$	$\forall x : \tau . M$	(Type, Prop)
Proof-terms	t, u	$::=$	ξ	
		$\mid$	$\lambda x . t \mid tu$	(Prop, Prop)

- (Type, Prop)-abstraction/application can be erased
- We can erase $\Lambda\alpha . M$ + $M\tau$ + type in $\lambda x : \tau . M$...

Why is system U^- inconsistent?

Kinds	τ, σ	$::=$	$\mathsf{Prop} \mid \alpha$	
		$\mid$	$\tau \to \sigma$	(Type, Type)
		$\mid$	$\Pi\alpha : \mathsf{Type} \,.\, \tau$	(Kind, Type)
Constructors	M, N	$::=$	ξ	
		$\mid$	$\lambda x \,.\, M \mid MN$	(Type, Type)
		$\mid$		(Kind, Type)
		$\mid$	$M \Rightarrow N$	(Prop, Prop)
		$\mid$	$\forall x : \tau \,.\, M$	(Type, Prop)
Proof-terms	t, u	$::=$	ξ	
		$\mid$	$\lambda x \,.\, t \mid tu$	(Prop, Prop)

- (Type, Prop)-abstraction/application can be erased
- We can erase $\Lambda\alpha \,.\, M$ + $M\tau$ + type in $\lambda x : \tau \,.\, M$...
  ... but makes no sense to remove τ in $\forall x : \tau \,.\, M$

Why is system U^- inconsistent?

Kinds	τ, σ	$::=$	$\mathsf{Prop} \mid \alpha$	
		$\mid$	$\tau \to \sigma$	(Type, Type)
		$\mid$	$\Pi\alpha : \mathsf{Type} \,.\, \tau$	(Kind, Type)
Constructors	M, N	$::=$	ξ	
		$\mid$	$\lambda x \,.\, M \mid MN$	(Type, Type)
		$\mid$		(Kind, Type)
		$\mid$	$M \Rightarrow N$	(Prop, Prop)
		$\mid$	$\forall x : \tau \,.\, M$	(Type, Prop)
Proof-terms	t, u	$::=$	ξ	
		$\mid$	$\lambda x \,.\, t \mid tu$	(Prop, Prop)

- (Type, Prop)-abstraction/application can be erased
- We can erase $\Lambda\alpha \,.\, M$ + $M\tau$ + type in $\lambda x : \tau \,.\, M$...
  ... but makes no sense to remove τ in $\forall x : \tau \,.\, M$
  Would identify propositions $\forall x, y : \mathsf{Unit} \,.\, x = y$ with $\forall x, y : \mathsf{Bool} \,.\, x = y$

Why is system U^- inconsistent?

Kinds	τ, σ	$::=$	$\mathsf{Prop} \mid \alpha$	
		$\mid$	$\tau \to \sigma$	(Type, Type)
		$\mid$	$\Pi\alpha : \mathsf{Type} \,.\, \tau$	(Kind, Type)
Constructors	M, N	$::=$	ξ	
		$\mid$	$\lambda x \,.\, M \mid MN$	(Type, Type)
		$\mid$		(Kind, Type)
		$\mid$	$M \Rightarrow N$	(Prop, Prop)
		$\mid$	$\forall x : \tau \,.\, M$	(Type, Prop)
Proof-terms	t, u	$::=$	ξ	
		$\mid$	$\lambda x \,.\, t \mid tu$	(Prop, Prop)

- (Type, Prop)-abstraction/application can be erased
- We can erase $\Lambda\alpha \,.\, M$ + $M\tau$ + type in $\lambda x : \tau \,.\, M$...
  ... but makes no sense to remove τ in $\forall x : \tau \,.\, M$
  Would identify propositions $\forall x, y : \mathsf{Unit} \,.\, x = y$ with $\forall x, y : \mathsf{Bool} \,.\, x = y$

$\Rightarrow$ (Kind, Type)-impredicativity is not parametric
i.e. cannot be reduced to an intersection