Equality and Isomorphisms in Type Theory

HoTT from scratch

Abdul Haliq

2024-11-07

AARHUS UNIVERSITET

Motivation

- **Types** in programming are a "compiler enforced" discipline to ensure well formed programs. i.e. $1 + 2$ would be legal unlike $1 + "a"$

Motivation

- **Types** in programming are a “compiler enforced” discipline to ensure well formed programs. i.e. $1 + 2$ would be legal unlike $1 + "a"$
- **Strong Types** with features such as generics are able to express more general and also more specific guarantees i.e. vector of some type variable X, vector of ints vs vector of untyped values

Motivation

- **Types** in programming are a “compiler enforced” discipline to ensure well formed programs. i.e. $1 + 2$ would be legal unlike $1 + "a"$
- **Strong Types** with features such as generics are able to express more general and also more specific guarantees i.e. vector of some type variable X, vector of ints vs vector of untyped values
- **Dependent Types** gives us a framework to express mathematics where terms are proofs and types are propositions. This automates verification of mathematical proofs and results, instead of relying on human experts. We can also now do formal methods that ensure correctness of programs rigorously using types.

Motivation

- **Types** in programming are a "compiler enforced" discipline to ensure well formed programs. i.e. $1 + 2$ would be legal unlike $1 + "a"$
- **Strong Types** with features such as generics are able to express more general and also more specific guarantees i.e. vector of some type variable X, vector of ints vs vector of untyped values
- **Dependent Types** gives us a framework to express mathematics where terms are proofs and types are propositions. This automates verification of mathematical proofs and results, instead of relying on human experts. We can also now do formal methods that ensure correctness of programs rigorously using types.
- **Homotopy Type Theory** and univalence asserts that when types are equal we automatically have a map for terms from one type to the other. Thus given a proof / term in one type, for free we get the proof in the other; proof transfer

Motivation

- **Types** in programming are a "compiler enforced" discipline to ensure well formed programs. i.e. $1 + 2$ would be legal unlike $1 + "a"$
- **Strong Types** with features such as generics are able to express more general and also more specific guarantees i.e. vector of some type variable X, vector of ints vs vector of untyped values
- **Dependent Types** gives us a framework to express mathematics where terms are proofs and types are propositions. This automates verification of mathematical proofs and results, instead of relying on human experts. We can also now do formal methods that ensure correctness of programs rigorously using types.
- **Homotopy Type Theory** and univalence asserts that when types are equal we automatically have a map for terms from one type to the other. Thus given a proof / term in one type, for free we get the proof in the other; proof transfer
- **Higher Observational Type Theory**: is one attempt at making HoTT practical (on a computer) using ideas from logical relations and parametricity.

Outline

1. Substitution Calculus

- **Deductive System**: a collection of rules

1.1 Deductive Systems

- **Deductive System**: a collection of rules
- **Rule**: takes hypotheses and gives a conclusion

$$\frac{\mathcal{H}_1 \quad \mathcal{H}_2 \quad \dots \quad \mathcal{H}_n}{\mathcal{C}}$$

- **Deductive System**: a collection of rules
- **Rule**: takes hypotheses and gives a conclusion

$$\frac{\mathcal{H}_1 \quad \mathcal{H}_2 \quad \dots \quad \mathcal{H}_n}{\mathcal{C}}$$

- **Axiom**: a rule with no hypotheses

$$\frac{}{\mathcal{C}}$$

- **Deductive System**: a collection of rules
- **Rule**: takes hypotheses and gives a conclusion

$$\frac{\mathcal{H}_1 \quad \mathcal{H}_2 \quad \ldots \quad \mathcal{H}_n}{\mathcal{C}}$$

- **Axiom**: a rule with no hypotheses

$$\frac{}{\mathcal{C}}$$

- **Judgement**: hypotheses or conclusions

Substitution Calculus Judgements

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$			

Substitution Calculus Judgements

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$	$\Gamma \vdash A\ \mathbf{Ty}$		

Substitution Calculus Judgements

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$	$\Gamma \vdash A\ \mathbf{Ty}$	$\Gamma \vdash a : A$	

Substitution Calculus Judgements

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$	$\Gamma \vdash A\ \mathbf{Ty}$	$\Gamma \vdash a : A$	$\Delta \vdash \gamma : \Gamma$

Substitution Calculus Judgements

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$	$\Gamma \vdash A\ \mathbf{Ty}$	$\Gamma \vdash a : A$	$\Delta \vdash \gamma : \Gamma$

- **Context**: lists of dependent terms

$$a_1 : A_1$$

Substitution Calculus Judgements

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$	$\Gamma \vdash A\ \mathbf{Ty}$	$\Gamma \vdash a : A$	$\Delta \vdash \gamma : \Gamma$

- **Context**: lists of dependent terms

$$a_1 : A_1, a_2 : A_2(a_1)$$

SUBSTITUTION CALCULUS JUDGEMENTS

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$	$\Gamma \vdash A\ \mathbf{Ty}$	$\Gamma \vdash a : A$	$\Delta \vdash \gamma : \Gamma$

- **Context**: lists of dependent terms

$$a_1 : A_1, a_2 : A_2(a_1), a_3 : A_3(a_1, a_2)$$

Substitution Calculus Judgements

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$	$\Gamma \vdash A\ \mathbf{Ty}$	$\Gamma \vdash a : A$	$\Delta \vdash \gamma : \Gamma$

- **Context**: lists of dependent terms

$$a_1 : A_1, a_2 : A_2(a_1), a_3 : A_3(a_1, a_2), ...a_n : A_n(a_1, a_2, ..., a_{n-1})$$

Substitution Calculus Judgements

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$	$\Gamma \vdash A\ \mathbf{Ty}$	$\Gamma \vdash a : A$	$\Delta \vdash \gamma : \Gamma$

- **Context**: lists of dependent terms

$$A_1.A_2.A_3.\ \dots\ .A_n$$

Substitution Calculus Judgements

Context	Type	Term	Substitution
$\Gamma\ \mathbf{cx}$	$\Gamma \vdash A\ \mathbf{Ty}$	$\Gamma \vdash a : A$	$\Delta \vdash \gamma : \Gamma$

- **Context**: lists of dependent terms

$$A_1 . A_2 . A_3 . \ \dots\ . A_n$$

- **Substitutions**: shifts judgements from one context to another

$$\frac{\Gamma \vdash A\ \mathbf{Ty} \quad \Delta \vdash \gamma : \Gamma}{\Delta \vdash A[\gamma]\ \mathbf{Ty}}$$

Substitution Calculus Equality Judgements

Type	Term	Substitution
$\Gamma \vdash A = B\ \mathbf{Ty}$		

Substitution Calculus Equality Judgements

Type	Term	Substitution
$\Gamma \vdash A = B \ \mathbf{Ty}$	$\Gamma \vdash a = b : A$	

Substitution Calculus Equality Judgements

Type	Term	Substitution
$\Gamma \vdash A = B\ \mathbf{Ty}$	$\Gamma \vdash a = b : A$	$\Delta \vdash \gamma = \delta : \Gamma$

Substitution Calculus Equality Judgements

Type	Term	Substitution
$\Gamma \vdash A = B \ \mathbf{Ty}$	$\Gamma \vdash a = b : A$	$\Delta \vdash \gamma = \delta : \Gamma$

definitionally equal symbols can be replaced by each other

Substitution Calculus Equality Judgements

Type	Term	Substitution
$\Gamma \vdash A = B\ \mathbf{Ty}$	$\Gamma \vdash a = b : A$	$\Delta \vdash \gamma = \delta : \Gamma$

definitionally equal symbols can be replaced by each other

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash A = B\ \mathbf{Ty}}{\Gamma \vdash a : B}$$

Substitution Calculus Rules

γ Application	**term**		
Morphism			
Pullback			

$$\frac{\Gamma \vdash a : A \quad \Delta \vdash \gamma : \Gamma}{\Delta \vdash a[\gamma] : A[\gamma]}\text{substitution-term}$$

Substitution Calculus Rules

γ Application	term	**type**	
Morphism			
Pullback			

$$\frac{\Gamma \vdash A\ \mathbf{Ty} \quad \Delta \vdash \gamma : \Gamma}{\Delta \vdash A[\gamma]\ \mathbf{Ty}}\text{substitution-type}$$

SUBSTITUTION CALCULUS RULES

γ Application	term	type	
Morphism	**composition**		
Pullback			

$$\frac{\Gamma_1 \vdash \gamma : \Gamma_0 \quad \Gamma_2 \vdash \gamma' : \Gamma_1}{\Gamma_2 \vdash \gamma \circ \gamma' : \Gamma_0}\text{composition}$$

Substitution Calculus Rules

γ Application	term	type	
Morphism	composition	**identity**	
Pullback			

$$\frac{\Gamma\ \mathbf{cx}}{\Gamma \vdash \mathbf{id} : \Gamma}\,\text{identity}$$

Substitution Calculus Rules

γ Application	term	type	
Morphism	composition	**identity**	
Pullback			

$$\frac{\Delta \vdash \gamma : \Gamma}{\Delta \vdash \gamma \circ \mathbf{id} = \mathbf{id} \circ \gamma = \gamma : \Gamma}\text{unital}$$

SUBSTITUTION CALCULUS RULES

γ Application	term	type	
Morphism	composition	identity	**associativity**
Pullback			

$$\frac{\Gamma_1 \vdash \gamma_0 : \Gamma_0 \quad \Gamma_2 \vdash \gamma_1 : \Gamma_1 \quad \Gamma_3 \vdash \gamma_2 : \Gamma_2}{\Gamma_3 \vdash \gamma_0 \circ (\gamma_1 \circ \gamma_2) = (\gamma_0 \circ \gamma_1) \circ \gamma_2 : \Gamma_0}\text{associativity}$$

Substitution Calculus Rules

γ Application	term	type	
Morphism	composition	identity	associativity
Pullback	**weakening**		

$$\frac{\Gamma \vdash A\ \mathbf{Ty}}{\Gamma.A \vdash \mathbf{p} : \Gamma}\text{weakening}$$

think as adding terms to context

Substitution Calculus Rules

γ Application	term	type	
Morphism	composition	identity	associativity
Pullback	weakening	**variable**	

$$\frac{\Gamma \vdash A \ \mathbf{Ty}}{\Gamma . A \vdash \mathbf{q} : A[\mathbf{p}]}\text{variable}$$

think as debruijn indices at index number of $\mathbf{p}$

Substitution Calculus Rules

γ Application	term	type	
Morphism	composition	identity	associativity
Pullback	weakening	**variable**	

$$\frac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma.A \vdash B_1\ \mathbf{Ty}}{\Gamma.A.B_1 \vdash \mathbf{q}[\mathbf{p}] : A[\mathbf{p}^2]}\text{variable}$$

think as debruijn indices at index number of $\mathbf{p}$

Substitution Calculus Rules

γ Application	term	type	
Morphism	composition	identity	associativity
Pullback	weakening	**variable**	

$$\frac{\Gamma \vdash A \ \mathbf{Ty} \quad \Gamma.A \vdash B_1 \ \mathbf{Ty} \quad \Gamma.A.B_1 \vdash B_2 \ \mathbf{Ty}}{\Gamma.A.B_1.B_2 \vdash \mathbf{q}[\mathbf{p}^2] : A[\mathbf{p}^3]}$$

think as debruijn indices at index number of $\mathbf{p}$

Substitution Calculus Rules

γ Application	term	type	
Morphism	composition	identity	associativity
Pullback	weakening	**variable**	

$$\frac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma.A \vdash B_1\ \mathbf{Ty} \quad \ldots \quad \Gamma.A.B_1 \ldots B_{n-1} \vdash B_n\ \mathbf{Ty}}{\Gamma.A.B_1 \ldots B_n \vdash \mathbf{q}[\mathbf{p}^n] : A[\mathbf{p}^{n+1}]}$$

notationally informally we still write $a : A$

Substitution Calculus Rules

γ Application	term	type	
Morphism	composition	identity	associativity
Pullback	weakening	variable	**substitution extension**

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A\ \mathbf{Ty} \quad \Delta \vdash a : A[\gamma]}{\Delta \vdash \gamma.a : \Gamma.A}\text{substitution-extension}$$

think removing / dispensing terms from context *the rules are justified / motivated by the natural model for dependent types*

$$\begin{tikzcd}
\Delta \arrow[r, dashed, "\Delta\vdash\gamma.a:\Gamma.A"] \arrow[rr, bend left, "\Delta\vdash a:A[\gamma]"] \arrow[dr, bend right, "\Delta\vdash\gamma:\Gamma"'] & \Gamma.A \arrow[r, "\Gamma.A\vdash\mathbf{q}:A[\mathbf{p}]"] \arrow[d, "\Gamma.A\vdash\mathbf{p}:\Gamma"'] & \mathbf{Tm} \arrow[d, "p"'] \\
& \Gamma \arrow[r, "\Gamma\vdash A\ \mathbf{Ty}"] & \mathbf{Ty}
\end{tikzcd}$$

we notate $\mathbf{Tm}(\Gamma, A)$ *for set of terms and* $\mathbf{Ty}(\Gamma)$ *for set of types*

2. Mapping In Types

2.1 Generally

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Upsilon(X)) \cong Y$$

- Type Υ is defined by the isomorphism with some set Y

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Upsilon(X)) \cong Y$$

- Type Υ is defined by the isomorphism with some set Y
- **Type Constructor**: $\Upsilon : (X : H) \to \mathbf{Ty}(\Gamma)$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Upsilon(X)) \cong Y$$

- Type Υ is defined by the isomorphism with some set Y
- **Type Constructor**: $\Upsilon : (X : H) \to \mathbf{Ty}(\Gamma)$

Formation	Υ

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Upsilon(X)) \cong Y$$

- Type Υ is defined by the isomorphism with some set Y
- **Type Constructor**: $\Upsilon : (X : H) \to \mathbf{Ty}(\Gamma)$

Formation	Υ
Introduction	ι_Γ^{-1}

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Upsilon(X)) \cong Y$$

- Type Υ is defined by the isomorphism with some set Y
- **Type Constructor**: $\Upsilon : (X : H) \to \mathbf{Ty}(\Gamma)$

Formation	Υ
Introduction	ι_Γ^{-1}
Elimination	ι_Γ

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Upsilon(X)) \cong Y$$

- Type Υ is defined by the isomorphism with some set Y
- **Type Constructor**: $\Upsilon : (X : H) \to \mathbf{Ty}(\Gamma)$

Formation	Υ
Introduction	ι_Γ^{-1}
Elimination	ι_Γ
Computation / β	$\iota_\Gamma \circ \iota_\Gamma^{-1} = \mathrm{id}$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Upsilon(X)) \cong Y$$

- Type Υ is defined by the isomorphism with some set Y
- **Type Constructor**: $\Upsilon : (X : H) \to \mathbf{Ty}(\Gamma)$

Formation	Υ
Introduction	ι_Γ^{-1}
Elimination	ι_Γ
Computation / β	$\iota_\Gamma \circ \iota_\Gamma^{-1} = \mathrm{id}$
Uniqueness / η	$\iota_\Gamma^{-1} \circ \iota_\Gamma = \mathrm{id}$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Pi(A, B)) \cong \mathbf{Tm}(\Gamma.A, B)$$

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma.A \vdash B\ \mathbf{Ty}}{\Gamma \vdash \Pi(A, B)\ \mathbf{Ty}}$ Π-formation

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Pi(A, B)) \cong \mathbf{Tm}(\Gamma.A, B)$$

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma.A \vdash B\ \mathbf{Ty}}{\Gamma \vdash \Pi(A, B)\ \mathbf{Ty}}\Pi\text{-formation}$
Introduction	$\dfrac{\Gamma.A \vdash b : B}{\Gamma \vdash \lambda(b) : \Pi(A, B)}\Pi\text{-intro}$

2.2 Π

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Pi(A, B)) \cong \mathbf{Tm}(\Gamma.A, B)$$

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma.A \vdash B\ \mathbf{Ty}}{\Gamma \vdash \Pi(A, B)\ \mathbf{Ty}}\ \Pi\text{-formation}$
Introduction	$\dfrac{\Gamma.A \vdash b : B}{\Gamma \vdash \lambda(b) : \Pi(A, B)}\ \Pi\text{-intro}$
Elimination	$\dfrac{\Gamma \vdash a : A \quad \Gamma \vdash f : \Pi(A, B)}{\Gamma \vdash \mathbf{app}(f, a) : B[\mathrm{id}\,.a]}\ \Pi\text{-elim}$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Pi(A, B)) \cong \mathbf{Tm}(\Gamma.A, B)$$

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma.A \vdash B\ \mathbf{Ty}}{\Gamma \vdash \Pi(A, B)\ \mathbf{Ty}}\Pi\text{-formation}$
Introduction	$\dfrac{\Gamma.A \vdash b : B}{\Gamma \vdash \lambda(b) : \Pi(A, B)}\Pi\text{-intro}$
Elimination	$\dfrac{\Gamma \vdash a : A \quad \Gamma \vdash f : \Pi(A, B)}{\Gamma \vdash \mathbf{app}(f, a) : B[\mathrm{id}\,.a]}\Pi\text{-elim}$
Computation / β	$\dfrac{\Gamma \vdash a : A \qquad \Gamma.A \vdash b : B}{\mathbf{app}(\lambda(b), a) = b[\mathrm{id}\,.a] : B[\mathrm{id}\,.a]}\Pi - \beta$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Pi(A, B)) \cong \mathbf{Tm}(\Gamma.A, B)$$

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma.A \vdash B\ \mathbf{Ty}}{\Gamma \vdash \Pi(A, B)\ \mathbf{Ty}}\,\Pi\text{-formation}$
Introduction	$\dfrac{\Gamma.A \vdash b : B}{\Gamma \vdash \lambda(b) : \Pi(A, B)}\,\Pi\text{-intro}$
Elimination	$\dfrac{\Gamma \vdash a : A \quad \Gamma \vdash f : \Pi(A, B)}{\Gamma \vdash \mathbf{app}(f, a) : B[\mathrm{id}\,.a]}\,\Pi\text{-elim}$
Computation / β	$\dfrac{\Gamma \vdash a : A \qquad \Gamma.A \vdash b : B}{\mathbf{app}(\lambda(b), a) = b[\mathrm{id}\,.a] : B[\mathrm{id}\,.a]}\,\Pi - \beta$
Uniqueness / η	$\dfrac{\Gamma \vdash f : \Pi(A, B)}{\lambda(\mathbf{app}(f[\mathbf{p}], \mathbf{q})) = f : \Pi(A, B)}\,\Pi - \eta$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Sigma(A, B)) \cong \mathbf{\Sigma}_{a:\mathbf{Tm}(\Gamma, A)} \ \mathbf{Tm}(\Gamma, B[\mathrm{id}\,.a])$$

think of $\mathbf{\Sigma}$ *as metatheory* Σ*, thus set of dependent pairs*

Formation	$\dfrac{\Gamma \vdash A \ \mathbf{Ty} \quad \Gamma.A \vdash B \ \mathbf{Ty}}{\Gamma \vdash \Sigma(A, B) \ \mathbf{Ty}}\ \Sigma\text{-formation}$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Sigma(A, B)) \cong \mathbf{\Sigma}_{a:\mathbf{Tm}(\Gamma,A)} \ \mathbf{Tm}(\Gamma, B[\mathrm{id}\,.a])$$

think of $\mathbf{\Sigma}$ *as metatheory* Σ*, thus set of dependent pairs*

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma.A \vdash B\ \mathbf{Ty}}{\Gamma \vdash \Sigma(A, B)\ \mathbf{Ty}}\ \Sigma\text{-formation}$
Introduction	$\dfrac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[\mathrm{id}\,.a]}{\Gamma \vdash \mathbf{pair}(a, b) : \Sigma(A, B)}\ \Sigma\text{-intro}$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Sigma(A, B)) \cong \mathbf{\Sigma}_{a:\mathbf{Tm}(\Gamma,A)}\ \mathbf{Tm}(\Gamma, B[\mathrm{id}\,.a])$$

think of $\mathbf{\Sigma}$ *as metatheory* Σ*, thus set of dependent pairs*

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma.A \vdash B\ \mathbf{Ty}}{\Gamma \vdash \Sigma(A, B)\ \mathbf{Ty}}\ \Sigma\text{-formation}$
Introduction	$\dfrac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[\mathrm{id}\,.a]}{\Gamma \vdash \mathbf{pair}(a, b) : \Sigma(A, B)}\ \Sigma\text{-intro}$
Elimination 1	$\dfrac{\Gamma \vdash p : \Sigma(A, B)}{\Gamma \vdash \mathbf{fst}(p) : A}\ \Sigma\text{-elim}_1$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \Sigma(A, B)) \cong \mathbf{\Sigma}_{a:\mathbf{Tm}(\Gamma, A)} \ \mathbf{Tm}(\Gamma, B[\mathrm{id} \, .a])$$

think of $\mathbf{\Sigma}$ *as metatheory* Σ, *thus set of dependent pairs*

Formation	$\dfrac{\Gamma \vdash A \ \mathbf{Ty} \quad \Gamma.A \vdash B \ \mathbf{Ty}}{\Gamma \vdash \Sigma(A, B) \ \mathbf{Ty}}\,\Sigma\text{-formation}$
Introduction	$\dfrac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[\mathrm{id} \, .a]}{\Gamma \vdash \mathbf{pair}(a, b) : \Sigma(A, B)}\,\Sigma\text{-intro}$
Elimination 1	$\dfrac{\Gamma \vdash p : \Sigma(A, B)}{\Gamma \vdash \mathbf{fst}(p) : A}\,\Sigma\text{-elim}_1$
Elimination 2	$\dfrac{\Gamma \vdash p : \Sigma(A, B)}{\Gamma \vdash \mathbf{snd}(p) : B[\mathrm{id} \, . \, \mathbf{fst}(p)]}\,\Sigma\text{-elim}_2$

2.4 Unit

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \mathbf{Unit}) \cong \{\star\}$$

Formation	$\dfrac{}{\Gamma \vdash \mathbf{Unit}\ \mathbf{Ty}}$ Unit-formation

2.4 Unit

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \mathbf{Unit}) \cong \{\star\}$$

Formation	$$\dfrac{}{\Gamma \vdash \mathbf{Unit}\ \mathbf{Ty}}\,\text{Unit-formation}$$
Introduction	$$\dfrac{\Gamma \vdash \mathbf{Unit}\ \mathbf{Ty}}{\Gamma \vdash \mathbf{tt} : \mathbf{Unit}}\,\text{Unit-intro}$$

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \mathbf{Unit}) \cong \{\star\}$$

Formation	$\dfrac{}{\Gamma \vdash \mathbf{Unit}\ \mathbf{Ty}}$ Unit-formation
Introduction	$\dfrac{\Gamma \vdash \mathbf{Unit}\ \mathbf{Ty}}{\Gamma \vdash \mathbf{tt} : \mathbf{Unit}}$ Unit-intro
Elimination	*what about the elimination rule?*

we will see in mapping out types

these isomorphisms respect substitution

$$\begin{tikzcd}
\mathbf{Tm}(\Gamma, \Upsilon(X)) \arrow[r, leftrightarrow, "\iota_\Gamma"] \arrow[d, "\gamma^*"'] & Y \arrow[d, "\psi_\gamma^*"'] \\
\mathbf{Tm}(\Delta, \Upsilon(X[\gamma])) \arrow[r, leftrightarrow, "\iota_\Delta"] & Y[\gamma]
\end{tikzcd}$$

we have to define rules for these as well for the types and terms of Υ

but we omit them in this presentation

2.6 Extensional Equality

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \mathbf{Eq}(A, a, b)) \cong \{\star \mid a = b\}$$

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash \mathbf{Eq}(A, a, b)\ \mathbf{Ty}}$ Eq-formation

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \mathbf{Eq}(A, a, b)) \cong \{\star \mid a = b\}$$

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash \mathbf{Eq}(A, a, b)\ \mathbf{Ty}}$ Eq-formation
Introduction	$\dfrac{\Gamma \vdash a : A}{\Gamma \vdash \mathbf{refl} : \mathbf{Eq}(A, a, a)}$ Eq-intro

$$\iota_\Gamma : \mathbf{Tm}(\Gamma, \mathbf{Eq}(A, a, b)) \cong \{\star \mid a = b\}$$

Formation	$\dfrac{\Gamma \vdash A\ \mathbf{Ty} \quad \Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash \mathbf{Eq}(A, a, b)\ \mathbf{Ty}}$ Eq-formation
Introduction	$\dfrac{\Gamma \vdash a : A}{\Gamma \vdash \mathbf{refl} : \mathbf{Eq}(A, a, a)}$ Eq-intro
Elimination	$\dfrac{\Gamma \vdash p : \mathbf{Eq}(A, a, b)}{\Gamma \vdash a = b : A}$ Eq-reflection

$$\frac{\Gamma \vdash p : \mathbf{Eq}(A, a, b)}{\Gamma \vdash a = b : A}\text{Eq-reflection}$$

Notice how elimination concludes a definitional equality judgement of terms rather than a term judgement. This departs from the usual elimination rules we have seen before.

$$\frac{\Gamma \vdash p : \mathbf{Eq}(A, a, b)}{\Gamma \vdash a = b : A}\text{Eq-reflection}$$

Notice how elimination concludes a definitional equality judgement of terms rather than a term judgement. This departs from the usual elimination rules we have seen before.

- **Propositional Equality**: equality is **internalized** within the theory e.g.

$$a = b : A \downarrow p : \mathbf{Eq}(A, a, b) \uparrow a = b : A$$

$$\frac{\Gamma \vdash p : \mathbf{Eq}(A, a, b)}{\Gamma \vdash a = b : A}\text{Eq-reflection}$$

Notice how elimination concludes a definitional equality judgement of terms rather than a term judgement. This departs from the usual elimination rules we have seen before.

- **Propositional Equality**: equality is **internalized** within the theory e.g.

$$a = b : A \downarrow p : \mathbf{Eq}(A, a, b) \uparrow a = b : A$$

- if we have a proof of equality, we can swap terms (definitionally)

$$\frac{\Gamma \vdash p : \mathbf{Eq}(A, a, b)}{\Gamma \vdash a = b : A}\text{Eq-reflection}$$

Notice how elimination concludes a definitional equality judgement of terms rather than a term judgement. This departs from the usual elimination rules we have seen before.

- **Propositional Equality**: equality is **internalized** within the theory e.g.

$$a = b : A \downarrow p : \mathbf{Eq}(A, a, b) \uparrow a = b : A$$

- if we have a proof of equality, we can swap terms (definitionally)
- e.g. with $p : \mathbf{Eq}(\mathbb{N}, a + b, b + a)$ we can swap $a + b$ for $b + a$ in lets say an argument for length of a vector without having to evaluate a and b

- **Uniqueness of Identity Proofs (UIP)**: there is only one proof of equality between two terms due to equality reflection, i.e. $p, q : \mathbf{Eq}(A, a, b)$ means $p = q$

- **Uniqueness of Identity Proofs (UIP)**: there is only one proof of equality between two terms due to equality reflection, i.e. $p, q : \mathbf{Eq}(A, a, b)$ means $p = q$
- **Injectivity**: distinct terms are mapped to distinct terms. But in UIP, all proofs of equality are mapped to $\star$ i.e. $\Gamma \vdash a = b : A$

- **Uniqueness of Identity Proofs (UIP)**: there is only one proof of equality between two terms due to equality reflection, i.e. $p, q : \mathbf{Eq}(A, a, b)$ means $p = q$
- **Injectivity**: distinct terms are mapped to distinct terms. But in UIP, all proofs of equality are mapped to $\star$ i.e. $\Gamma \vdash a = b : A$
- **Normalization**: with UIP our types aren't injective, without injective types we can't have normalization

- **Uniqueness of Identity Proofs (UIP)**: there is only one proof of equality between two terms due to equality reflection, i.e. $p, q : \mathbf{Eq}(A, a, b)$ means $p = q$
- **Injectivity**: distinct terms are mapped to distinct terms. But in UIP, all proofs of equality are mapped to $\star$ i.e. $\Gamma \vdash a = b : A$
- **Normalization**: with UIP our types aren't injective, without injective types we can't have normalization (we won't explore why here, but lets take it for granted for now)

- **Uniqueness of Identity Proofs (UIP)**: there is only one proof of equality between two terms due to equality reflection, i.e. $p, q : \mathbf{Eq}(A, a, b)$ means $p = q$
- **Injectivity**: distinct terms are mapped to distinct terms. But in UIP, all proofs of equality are mapped to $\star$ i.e. $\Gamma \vdash a = b : A$
- **Normalization**: with UIP our types aren't injective, without injective types we can't have normalization (we won't explore why here, but lets take it for granted for now)

Without normalization we can't have decidable type checking!

- **Uniqueness of Identity Proofs (UIP)**: there is only one proof of equality between two terms due to equality reflection, i.e. $p, q : \mathbf{Eq}(A, a, b)$ means $p = q$
- **Injectivity**: distinct terms are mapped to distinct terms. But in UIP, all proofs of equality are mapped to $\star$ i.e. $\Gamma \vdash a = b : A$
- **Normalization**: with UIP our types aren't injective, without injective types we can't have normalization (we won't explore why here, but lets take it for granted for now)

Without normalization we can't have decidable type checking!

We must do propositional equality differently

3. Mapping Out Types

$$\{c \in \mathbf{Tm}(\Gamma.\Upsilon, C) \mid \mathbf{rec}\} \cong \{\star\}$$

we define Υ by the terms it maps out to in C, hence the name

$$\{c \in \mathbf{Tm}(\Gamma.\Upsilon, C) \mid \mathbf{rec}\} \cong \{\star\}$$

we define Υ by the terms it maps out to in C, hence the name

$\Upsilon(X)$	signature	initial algebra
Void	$X \mapsto 0$	absurd

$$\{c \in \mathbf{Tm}(\Gamma.\Upsilon, C) \mid \mathbf{rec}\} \cong \{\star\}$$

we define Υ *by the terms it maps out to in* C*, hence the name*

$\Upsilon(X)$	signature	initial algebra
$\mathbf{Void}$	$X \mapsto 0$	absurd
$\mathbf{Unit}$	$X \mapsto 1$	$\mathbf{tt}$

$$\{c \in \mathbf{Tm}(\Gamma.\Upsilon, C) \mid \mathbf{rec}\} \cong \{\star\}$$

we define Υ by the terms it maps out to in C, hence the name

$\Upsilon(X)$	signature	initial algebra
Void	$X \mapsto 0$	absurd
Unit	$X \mapsto 1$	**tt**
Bool	$X \mapsto 1 + 1$	**true, false**

$$\{c \in \mathbf{Tm}(\Gamma.\Upsilon, C) \mid \mathbf{rec}\} \cong \{\star\}$$

we define Υ by the terms it maps out to in C, hence the name

$\Upsilon(X)$	signature	initial algebra
Void	$X \mapsto 0$	absurd
Unit	$X \mapsto 1$	**tt**
Bool	$X \mapsto 1+1$	**true**, **false**
$A+B$	$X \mapsto A+B$	inl, inr

$$\{c \in \mathbf{Tm}(\Gamma.\Upsilon, C) \mid \mathbf{rec}\} \cong \{\star\}$$

we define Υ by the terms it maps out to in C, hence the name

$\Upsilon(X)$	signature	initial algebra
$\mathbf{Void}$	$X \mapsto 0$	absurd
$\mathbf{Unit}$	$X \mapsto 1$	$\mathbf{tt}$
$\mathbf{Bool}$	$X \mapsto 1 + 1$	$\mathbf{true}, \mathbf{false}$
$A + B$	$X \mapsto A + B$	inl, inr
$\mathbb{N}$	$X \mapsto 1 + X$	zero, succ

$$\{c \in \mathbf{Tm}(\Gamma.\Upsilon, C) \mid \mathbf{rec}\} \cong \{\star\}$$

we define Υ by the terms it maps out to in C, hence the name

$\Upsilon(X)$	signature	initial algebra
Void	$X \mapsto 0$	absurd
Unit	$X \mapsto 1$	**tt**
Bool	$X \mapsto 1 + 1$	**true**, **false**
$A + B$	$X \mapsto A + B$	inl, inr
$\mathbb{N}$	$X \mapsto 1 + X$	zero, succ

- **Formation Rule**: $\Upsilon(X)$
- **Intro**: initial algebra constructors
- **Elim**: **rec**

3.2 Unit

the **Unit** *type now defined as a mapping out type*

$$\{c \in \mathbf{Tm}(\Gamma.\ \mathbf{Unit}, C) \mid c = c[\mathrm{id} \,.\, \mathbf{tt}]\} \cong \{\star\}$$

3.2 **Unit**

the **Unit** *type now defined as a mapping out type*

$$\{c \in \mathbf{Tm}(\Gamma.\ \mathbf{Unit}, C) \mid c = c[\mathrm{id}\,.\ \mathbf{tt}]\} \cong \{\star\}$$

$$\mathbf{rec}(\mathbf{tt}, c) = c : C[\mathrm{id}\,.\ \mathbf{tt}]$$

the **Unit** *type now defined as a mapping out type*

$$\{c \in \mathbf{Tm}(\Gamma.\ \mathbf{Unit}, C) \mid c = c[\mathrm{id}\,.\ \mathbf{tt}]\} \cong \{\star\}$$

$$\mathbf{rec}(\mathbf{tt}, c) = c : C[\mathrm{id}\,.\ \mathbf{tt}]$$

$$\frac{\Gamma \vdash u : \mathbf{Unit} \quad \Gamma \vdash c : C[\mathrm{id}\,.\ \mathbf{tt}]}{\Gamma \vdash \mathbf{rec}(u, c) : C[\mathrm{id}\,.u]}\text{Unit-elim}$$

the **Bool** *type is a mapping out type*

$$\{c \in \mathbf{Tm}(\Gamma.\,\mathbf{Bool}, C) \mid c_{\mathbf{true}} = c[\mathrm{id}\,.\,\mathbf{true}] \wedge c_{\mathbf{false}} = c[\mathrm{id}\,.\,\mathbf{false}]\} \cong \{\star\}$$

the **Bool** *type is a mapping out type*

$$\{c \in \mathbf{Tm}(\Gamma.\ \mathbf{Bool}, C) \mid c_{\mathbf{true}} = c[\mathrm{id}\ .\ \mathbf{true}] \wedge c_{\mathbf{false}} = c[\mathrm{id}\ .\ \mathbf{false}]\} \cong \{\star\}$$

$$\mathbf{rec}(\mathbf{true}, c_{\mathbf{true}}, c_{\mathbf{false}}) = c_{\mathbf{true}} : C[\mathrm{id}\ .\ \mathbf{true}]$$

$$\mathbf{rec}(\mathbf{false}, c_{\mathbf{true}}, c_{\mathbf{false}}) = c_{\mathbf{false}} : C[\mathrm{id}\ .\ \mathbf{false}]$$

the **Bool** *type is a mapping out type*

$$\{c \in \mathbf{Tm}(\Gamma .\ \mathbf{Bool}, C) \mid c_{\mathbf{true}} = c[\mathrm{id} .\ \mathbf{true}] \wedge c_{\mathbf{false}} = c[\mathrm{id} .\ \mathbf{false}]\} \cong \{\star\}$$

$$\mathbf{rec}(\mathbf{true}, c_{\mathbf{true}}, c_{\mathbf{false}}) = c_{\mathbf{true}} : C[\mathrm{id} .\ \mathbf{true}]$$
$$\mathbf{rec}(\mathbf{false}, c_{\mathbf{true}}, c_{\mathbf{false}}) = c_{\mathbf{false}} : C[\mathrm{id} .\ \mathbf{false}]$$

$$\frac{\Gamma \vdash b : \mathbf{Bool} \quad \Gamma \vdash c_{\mathbf{true}} : C[\mathrm{id} .\ \mathbf{true}] \quad \Gamma \vdash c_{\mathbf{false}} : C[\mathrm{id} .\ \mathbf{false}]}{\Gamma \vdash \mathbf{rec}(b, c_{\mathbf{true}}, c_{\mathbf{false}}) : C[\mathrm{id} . b]}\ \text{Bool-elim}$$

the **Bool** *type is a mapping out type*

$$\{c \in \mathbf{Tm}(\Gamma.\ \mathbf{Bool}, C) \mid c_{\mathbf{true}} = c[\mathrm{id}\ .\ \mathbf{true}] \wedge c_{\mathbf{false}} = c[\mathrm{id}\ .\ \mathbf{false}]\} \cong \{\star\}$$

$$\mathbf{rec}(\mathbf{true}, c_{\mathbf{true}}, c_{\mathbf{false}}) = c_{\mathbf{true}} : C[\mathrm{id}\ .\ \mathbf{true}]$$

$$\mathbf{rec}(\mathbf{false}, c_{\mathbf{true}}, c_{\mathbf{false}}) = c_{\mathbf{false}} : C[\mathrm{id}\ .\ \mathbf{false}]$$

$$\frac{\Gamma \vdash b : \mathbf{Bool} \quad \Gamma \vdash c_{\mathbf{true}} : C[\mathrm{id}\ .\ \mathbf{true}] \quad \Gamma \vdash c_{\mathbf{false}} : C[\mathrm{id}\ .\ \mathbf{false}]}{\Gamma \vdash \mathbf{rec}(b, c_{\mathbf{true}}, c_{\mathbf{false}}) : C[\mathrm{id}\ .b]}\,\text{Bool-elim}$$

for **Bool** *often* **rec** *is also written as* **if**

3.4 $A + B$

the disjoint sum type; $A + B$*, is like a* $\mathbf{Bool}$ *with arguments*

$$\{c \in \mathbf{Tm}(\Gamma.A + B, C) \mid c_A = c[\mathrm{id}\,.\ \mathrm{inl}\ a] \wedge c_B = c[\mathrm{id}\,.\ \mathrm{inr}\ b]\} \cong \{\star\}$$

$$\mathbf{rec}(\mathrm{inl}\ a, c_A, c_B) = c_A : C[\mathrm{id}\,.\ \mathrm{inl}\ a]$$
$$\mathbf{rec}(\mathrm{inr}\ b, c_A, c_B) = c_B : C[\mathrm{id}\,.\ \mathrm{inr}\ b]$$

$$\frac{\Gamma \vdash o : A + B \quad \Gamma \vdash c_A : C[\mathrm{id}\,.\ \mathrm{inl}\ a] \quad \Gamma \vdash c_B : C[\mathrm{id}\,.\ \mathrm{inr}\ b]}{\Gamma \vdash \mathbf{rec}(o, c_A, c_B) : C[\mathrm{id}\,.o]}\ A + B\text{-elim}$$

$\mathbb{N}$ *motivates why we call the elimination rule a recursor* **rec**

$$\{c \in \mathbf{Tm}(\Gamma.\mathbb{N}, C) \mid c_z = c[\mathrm{id}\,.\,\mathrm{zero}] \wedge c_s[\mathbf{p}.\mathbf{q}.c] = c[\mathbf{p}.\,\mathrm{succ}(\mathbf{q})]\} \cong \{\star\}$$

$$\mathbf{rec}(\mathrm{zero}, c_z, c_s) = c_z : C[\mathrm{id}\,.\,\mathrm{zero}]$$
$$\mathbf{rec}(\mathrm{succ}(n), c_z, c_s) = c_s[\mathrm{id}\,.n.\,\mathbf{rec}(n, c_z, c_s)] : C[\mathrm{id}\,.\,\mathrm{succ}(n)]$$

$$\frac{\Gamma \vdash n : \mathbb{N} \quad \Gamma.\mathbb{N} \vdash C\ \mathbf{Ty} \quad \Gamma \vdash c_z : C[\mathrm{id}\,.\,\mathrm{zero}] \quad \Gamma.\mathbb{N}.C \vdash c_s : C[\mathbf{p}^2.\,\mathrm{succ}(\mathbf{q}[\mathbf{p}])]}{\Gamma \vdash \mathbf{rec}(n, c_z, c_s) : C[\mathrm{id}\,.n]}\ \mathbb{N}\text{-elim}$$

3.6 **Void**

$\mathbf{Void}$ *has no recursor arguments in* C

$$\mathbf{Tm}(\Gamma.\ \mathbf{Void}, C) \cong \{\star\}$$

$\mathbf{rec}(v) : C[\mathrm{id}\,.v]$ *often written as* $\mathrm{absurd}(v) : C[\mathrm{id}\,.v]$

$$\frac{\Gamma \vdash v : \mathbf{Void} \quad \Gamma.\ \mathbf{Void} \vdash C\ \mathbf{Ty}}{\Gamma \vdash \mathrm{absurd}(v) : C[\mathrm{id}\,.v]}\ \text{Void-elim}$$

notice how ***ANY*** *term in* C *that depends on* $\mathbf{Void}$ *is uniquely* $\mathrm{absurd}(v)$

4. Intensional Equality

We now try to define Propositional Equality as a mapping out type

$$\{c \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(A, a, b), C) \mid \mathbf{J}\} \cong \{\star\}$$

- **Introduction Rule**: still the same $\mathbf{refl} : \mathbf{Id}(A, a, a)$
- **Elimination Rule**: $\mathbf{J}$ when presented as a "functional program" is as follows:

$$\begin{aligned} &\mathbf{J} : \{A : \mathbf{U}\} \\ &\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \end{aligned}$$

We now try to define Propositional Equality as a mapping out type

$$\{c \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(A, a, b), C) \mid \mathbf{J}\} \cong \{\star\}$$

- **Introduction Rule**: still the same $\mathbf{refl} : \mathbf{Id}(A, a, a)$
- **Elimination Rule**: $\mathbf{J}$ when presented as a "functional program" is as follows:

$$\begin{aligned}
&\mathbf{J} : \{A : \mathbf{U}\} \\
&\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\
&\to (a : A \to C(a, a, \mathbf{refl}_a))
\end{aligned}$$

We now try to define Propositional Equality as a mapping out type

$$\{c \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(A, a, b), C) \mid \mathbf{J}\} \cong \{\star\}$$

- **Introduction Rule**: still the same $\mathbf{refl} : \mathbf{Id}(A, a, a)$
- **Elimination Rule**: $\mathbf{J}$ when presented as a “functional program” is as follows:

$$\begin{aligned}
&\mathbf{J} : \{A : \mathbf{U}\} \\
&\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\
&\to (a : A \to C(a, a, \mathbf{refl}_a)) \\
&\to a, b : A \to p : \mathbf{Id}(A, a, b)
\end{aligned}$$

We now try to define Propositional Equality as a mapping out type

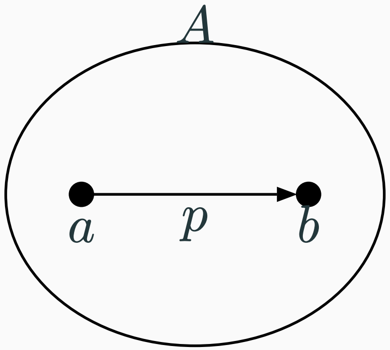
$$\{c \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(A, a, b), C) \mid \mathbf{J}\} \cong \{\star\}$$

- **Introduction Rule**: still the same $\mathbf{refl} : \mathbf{Id}(A, a, a)$
- **Elimination Rule**: $\mathbf{J}$ when presented as a "functional program" is as follows:

$$\begin{aligned}
&\mathbf{J} : \{A : \mathbf{U}\} \\
&\qquad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\
&\to (a : A \to C(a, a, \mathbf{refl}_a)) \\
&\to a, b : A \to p : \mathbf{Id}(A, a, b) \\
&\to C(a, b, p)
\end{aligned}$$

we will see why $\mathbf{J}$ *is justified to model equality as follows* *to aid visualization, we introduce the homotopy interpretation of types*

- types are spaces
- terms are points
- identifications are paths



$$\mathbf{J} \Rightarrow \text{subst}$$

$$\begin{aligned}
\mathbf{J} &: \{A : \mathbf{U}\} \\
&\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\
&\to (a : A \to C\ a\ a\ \mathbf{refl}_a) \\
&\to a, b : A \to p : \mathbf{Id}(A, a, b) \\
&\to C\ a\ b\ p
\end{aligned}$$

$$\text{subst}\ \{a, b : A\} B\ p = \mathbf{J}$$

4.3 subst

$$\mathbf{J} \Rightarrow \text{subst}$$

$$\begin{aligned}
\mathbf{J} &: \{A : \mathbf{U}\} \\
&\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\
&\to (a : A \to C\ a\ a\ \mathbf{refl}_a) \\
&\to a, b : A \to p : \mathbf{Id}(A, a, b) \\
&\to C\ a\ b\ p
\end{aligned}$$

$$\begin{aligned}
\text{subst}\ \{a, b : A\} B\ p = \mathbf{J} \\
\lambda a\ b\ _.(B\ a \to B\ b)
\end{aligned}$$

$$\mathbf{J} \Rightarrow \text{subst}$$

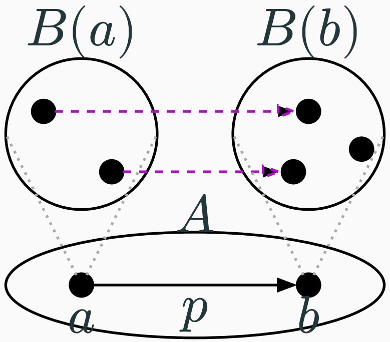
$$\begin{aligned}
&\mathbf{J} : \{A : \mathbf{U}\} \\
&\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\
&\to (a : A \to C\ a\ a\ \mathbf{refl}_a) \\
&\to a, b : A \to p : \mathbf{Id}(A, a, b) \\
&\to C\ a\ b\ p
\end{aligned}$$

$$\begin{aligned}
&\text{subst}\ \{a, b : A\} B\ p = \mathbf{J} \\
&\quad \lambda a\ b\ _.(B\ a \to B\ b) \\
&\quad \lambda_.\lambda b.b
\end{aligned}$$

$$\mathbf{J} \Rightarrow \text{subst}$$

$$\begin{aligned}
\mathbf{J} : & \{A : \mathbf{U}\} \\
& (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\
\to & (a : A \to C\ a\ a\ \mathbf{refl}_a) \\
\to & a, b : A \to p : \mathbf{Id}(A, a, b) \\
\to & C\ a\ b\ p
\end{aligned}$$

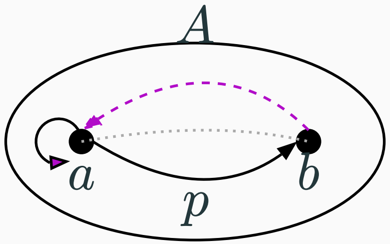
$$\begin{aligned}
\text{subst}\ & \{a, b : A\} B\ p = \mathbf{J} \\
& \lambda a\ b\ _.(B\ a \to B\ b) \\
& \lambda _.\lambda b.b \\
& a\ b\ p
\end{aligned}$$



think subst *gives a function that "transports" terms in Ba to Bb*

$$\mathbf{J} \Rightarrow \mathrm{subst} \Rightarrow \mathrm{sym}$$

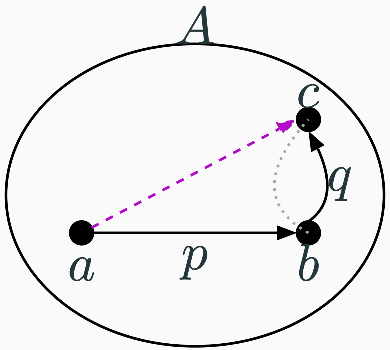
$$\begin{aligned} \mathrm{subst} : \{a, b : A\} \quad & \quad \mathrm{sym}\ p = \mathrm{subst} \\ B : A \to \mathbf{U} \quad & \quad \lambda x.\ \mathbf{Id}(A, x, a) \\ \to \mathbf{Id}(A, a, b) \quad & \quad p \\ \to B\ a \quad & \quad \mathbf{refl}_a \\ \to B\ b \quad & \end{aligned}$$



think sym *"drags" the origin of* $\mathbf{refl}_a$ *along* p *to get its "inverse"*

$$\mathbf{J} \Rightarrow \text{subst} \Rightarrow \text{trans}$$

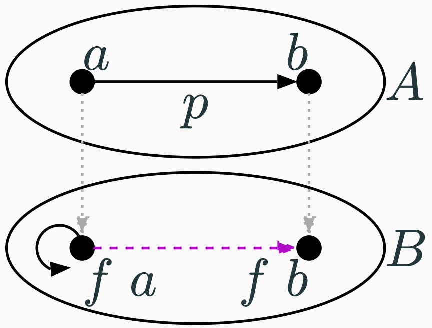
$$\begin{aligned}
\text{subst} : \{a, b : A\} \quad & \quad \text{trans } p\ q = \text{subst} \\
B : A \to \mathbf{U} \quad & \quad \lambda x.\ \mathbf{Id}(A, a, x) \\
\to \mathbf{Id}(A, a, b) \quad & \quad p \\
\to B\ a \quad & \quad q \\
\to B\ b \quad &
\end{aligned}$$



think trans *"drags" end of p along q to get the transitive*

$$\mathbf{J} \Rightarrow \text{subst} \Rightarrow \text{cong}$$

$$\begin{aligned} \text{subst} : \{a, b : A\} \\ B : A \to \mathbf{U} \\ \to \mathbf{Id}(A, a, b) \\ \to B\ a \\ \to B\ b \end{aligned} \qquad \begin{aligned} \text{cong}\ f\ p = \text{subst} \\ \lambda x.\ \mathbf{Id}(B, f\ a, f\ x) \\ p \\ \mathbf{refl}_{f\ a} \end{aligned}$$



think cong *"drags" end of* $\mathbf{refl}_{f\ a}$ *along a path in* B *"parallel" to* p

$$\mathbf{J} \Rightarrow \text{uniq}$$

$$\textit{let } [z] = \Sigma(z : A, \mathbf{Id}(A, x, z)); \textit{ points idetnfied with } z$$

$$\begin{aligned}
\mathbf{J} &: \{A : \mathbf{U}\} \\
&\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\
&\to (a : A \to C\ a\ a\ \mathbf{refl}_a) \\
&\to a, b : A \to p : \mathbf{Id}(A, a, b) \\
&\to C\ a\ b\ p
\end{aligned}$$

$$\text{uniq } \{a : A\} b\ p = \mathbf{J}$$

$$\mathbf{J} \Rightarrow \text{uniq}$$

$$let\ [z] = \Sigma(z : A, \mathbf{Id}(A, x, z)); \textit{points idetnfied with } z$$

$$\begin{aligned} \mathbf{J} &: \{A : \mathbf{U}\} \\ &\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\ &\to (a : A \to C\ a\ a\ \mathbf{refl}_a) \\ &\to a, b : A \to p : \mathbf{Id}(A, a, b) \\ &\to C\ a\ b\ p \end{aligned}$$

$$\begin{aligned} &\text{uniq}\ \{a : A\} b\ p = \mathbf{J} \\ &\quad \lambda x, y, p'.\ \mathbf{Id}([z], (x, \mathbf{refl}_x), (y, p')) \end{aligned}$$

$$\mathbf{J} \Rightarrow \mathrm{uniq}$$

$$\textit{let } [z] = \Sigma(z : A, \mathbf{Id}(A, x, z)); \textit{points idetnfied with } z$$

$$\begin{aligned}
&\mathbf{J} : \{A : \mathbf{U}\} \\
&\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma)) \\
&\to (a : A \to C\ a\ a\ \mathbf{refl}_a) \\
&\to a, b : A \to p : \mathbf{Id}(A, a, b) \\
&\to C\ a\ b\ p
\end{aligned}$$

$$\begin{aligned}
&\mathrm{uniq}\ \{a : A\} b\ p = \mathbf{J} \\
&\quad \lambda x, y, p'.\ \mathbf{Id}([z], (x, \mathbf{refl}_x), (y, p')) \\
&\quad \lambda x. \mathbf{refl}_{x, \mathbf{refl}_x}
\end{aligned}$$

$$\mathbf{J} \Rightarrow \text{uniq}$$

$$\textit{let } [z] = \Sigma(z : A, \mathbf{Id}(A, x, z)); \textit{ points idetnfied with } z$$

$\mathbf{J} : \{A : \mathbf{U}\}$

$\quad (C : (a, b : A) \to \mathbf{Id}(A, a, b) \to \mathbf{Ty}(\Gamma))$

$\to (a : A \to C\ a\ a\ \mathbf{refl}_a)$

$\to a, b : A \to p : \mathbf{Id}(A, a, b)$

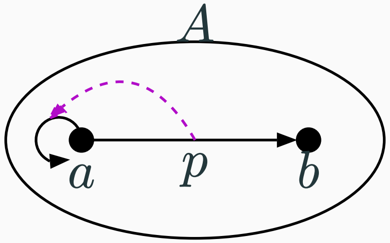
$\to C\ a\ b\ p$

$\text{uniq}\ \{a : A\} b\ p = \mathbf{J}$

$\quad \lambda x, y, p'.\ \mathbf{Id}([z], (x, \mathbf{refl}_x), (y, p'))$

$\quad \lambda x.\mathbf{refl}_{x, \mathbf{refl}_x}$

$\quad a\ b\ p$



think uniq *identifies all identities from* a *to* b *with* $\mathbf{refl}_a$

4.8 Metatheory of $\mathbf{Id}$

- **J is Independent**: $\mathbf{J}$ can still be defined in an $\mathbf{Eq}$ with reflection, but it will be a trivial structure due to UIP

- **J is Independent**: $\mathbf{J}$ can still be defined in an $\mathbf{Eq}$ with reflection, but it will be a trivial structure due to UIP
- **Function Extensionality (funext)**: We can't construct a term for such a type, assuming it as an axiom breaks canonicity (i.e. a bool term is judgementally neither true or false)

$$\text{Funext} = (A : \mathbf{U}) \to (B : A \to \mathbf{U}) \to (f, g : (a : A) \to B\ a)$$
$$\to ((a : A) \to \mathbf{Id}(B\ a, f\ a, g\ a)) \to \mathbf{Id}((a : A) \to B\ a, f, g)$$

4.8 Metatheory of **Id**

- **J is Independent**: $\mathbf{J}$ can still be defined in an $\mathbf{Eq}$ with reflection, but it will be a trivial structure due to UIP
- **Function Extensionality (funext)**: We can't construct a term for such a type, assuming it as an axiom breaks canonicity (i.e. a bool term is judgementally neither true or false)

$$\mathrm{Funext} = (A : \mathbf{U}) \to (B : A \to \mathbf{U}) \to (f, g : (a : A) \to B\ a) \\ \to ((a : A) \to \mathbf{Id}(B\ a, f\ a, g\ a)) \to \mathbf{Id}((a : A) \to B\ a, f, g)$$

- **No UIP**: We can't construct a term for such a type

$$\mathrm{UIP} = (A : \mathbf{U}) \to (a, b : A) \to (p, q : \mathbf{Id}(A, a, b)) \to \mathbf{Id}(\mathbf{Id}(A, a, b), p, q)$$

- **J is Independent**: **J** can still be defined in an **Eq** with reflection, but it will be a trivial structure due to UIP
- **Function Extensionality (funext)**: We can't construct a term for such a type, assuming it as an axiom breaks canonicity (i.e. a bool term is judgementally neither true or false)

$$\text{Funext} = (A : \mathbf{U}) \to (B : A \to \mathbf{U}) \to (f, g : (a : A) \to B\ a)$$
$$\to ((a : A) \to \mathbf{Id}(B\ a, f\ a, g\ a)) \to \mathbf{Id}((a : A) \to B\ a, f, g)$$

- **No UIP**: We can't construct a term for such a type

$$\text{UIP} = (A : \mathbf{U}) \to (a, b : A) \to (p, q : \mathbf{Id}(A, a, b)) \to \mathbf{Id}(\mathbf{Id}(A, a, b), p, q)$$

- **Axiom K**: but we can add a second eliminator that identifies identifications of a term to itself with refl to recover UIP whilst still having canonicity and normalization

4.8 Metatheory of **Id**

- **J is Independent**: **J** can still be defined in an **Eq** with reflection, but it will be a trivial structure due to UIP
- **Function Extensionality (funext)**: We can't construct a term for such a type, assuming it as an axiom breaks canonicity (i.e. a bool term is judgementally neither true or false)

$$\begin{aligned}\text{Funext} = (A : \mathbf{U}) \to (B : A \to \mathbf{U}) \to (f, g : (a : A) \to B\, a) \\ \to ((a : A) \to \mathbf{Id}(B\, a, f\, a, g\, a)) \to \mathbf{Id}((a : A) \to B\, a, f, g)\end{aligned}$$

- **No UIP**: We can't construct a term for such a type

$$\text{UIP} = (A : \mathbf{U}) \to (a, b : A) \to (p, q : \mathbf{Id}(A, a, b)) \to \mathbf{Id}(\mathbf{Id}(A, a, b), p, q)$$

- **Axiom K**: but we can add a second eliminator that identifies identifications of a term to itself with refl to recover UIP whilst still having canonicity and normalization
- **Hoffman Conservativity Theorem**: all propositions that are provable (construct terms for a type) in ETT but not in ITT boils down to funext and UIP

- **Full Spectrum**: To use types in our theory, we need a notion of a universe whose terms are types, thus we can omit the formation rules of Υ for introduction rules of $\mathbf{U}$

$$\begin{tikzcd} \mathbf{Tm}(\Gamma, \mathbf{U}) \arrow[rr, bend left, "\mathbf{El}"] & & \mathbf{Ty}(\Gamma) \arrow[ll, bend left, "\mathbf{code}"] \end{tikzcd}$$

- thus we have dependent version of **rec** called **ind** for inductive types now

- **Full Spectrum**: To use types in our theory, we need a notion of a universe whose terms are types, thus we can omit the formation rules of Υ for introduction rules of $\mathbf{U}$

$$\begin{tikzcd} \mathbf{Tm}(\Gamma, \mathbf{U}) \arrow[rr, bend left, "\mathbf{El}"] & & \mathbf{Ty}(\Gamma) \arrow[ll, bend left, "\mathbf{code}"] \end{tikzcd}$$

- thus we have dependent version of **rec** called **ind** for inductive types now
- **Recursive Types**: i.e. $\Pi(\mathbf{U}, -)$ can't be made without having a code for $\mathbf{U} : \mathbf{U}$, but doing so causes impredicative paradoxes making the theory inconsistent (we won't prove it here), thus we make a infinite hierarchy of cumulative universes

$$\frac{\Gamma \vdash c : \mathbf{U}_i}{\Gamma \vdash \mathbf{lift}_{i(c)} : \mathbf{U}_{i+1}}\text{Lifting} \qquad \frac{j < i \quad \Gamma \vdash \mathbf{U}_i \text{ type} \quad \Gamma \vdash \mathbf{U}_j \text{ type}}{\Gamma \vdash \mathbf{uni}_{i,j} : \mathbf{U}_i}\text{Universe}$$

5. Univalence

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(U, A, B), C) \mid \downarrow\} \cong \{\star\}$$

- we want C to work as a "bridge" that brings proofs in A to B

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(U, A, B), C) \mid \downarrow\} \cong \{\star\}$$

- we want C to work as a “bridge” that brings proofs in A to B

$$C\ \mathbf{Id}(U, A, B)\ (C\ A\ B)$$

- the universe is said to be univalent if the map from the identification to C is C as well

5.1 Generally

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(U, A, B), C) \mid \downarrow\} \cong \{\star\}$$

- we want C to work as a "bridge" that brings proofs in A to B

$$C\ \mathbf{Id}(U, A, B)\ (C\ A\ B)$$

- the universe is said to be univalent if the map from the identification to C is C as well
- naively we might consider the isomorphisms or bimaps, but this is too strong a condition that it only works on a subset of $\mathbf{U}$ called $\mathbf{HProp}$ which are homotopy propositions, types with only one term, but we will explore them to motivate the full definition

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{HProp}, A, B), \cong) \mid \mathrm{idtoiso}\} \cong \{\star\}$$

5.2 Prop Univalence

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{HProp}, A, B), \cong) \mid \mathrm{idtoiso}\} \cong \{\star\}$$

$$\left(A \underset{\mathbf{HProp}}{=} B\right) \cong (A \cong B)$$

5.2 Prop Univalence

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{HProp}, A, B), \cong) \mid \mathrm{idtoiso}\} \cong \{\star\}$$

$$\left(A \underset{\mathbf{HProp}}{=} B\right) \cong (A \cong B)$$

$$A \cong B = \Sigma((f, g) : A \leftrightarrow B, \mathrm{areIso}(f, g))$$

5.2 Prop Univalence

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{HProp}, A, B), \cong) \mid \mathrm{idtoiso}\} \cong \{\star\}$$

$$\left(A \underset{\mathbf{HProp}}{=} B\right) \cong (A \cong B)$$

$$A \cong B = \Sigma((f, g) : A \leftrightarrow B, \mathrm{areIso}(f, g))$$

$$A \leftrightarrow B = (A \to B) \times (B \to A)$$

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{HProp}, A, B), \cong) \mid \mathrm{idtoiso}\} \cong \{\star\}$$

$$\left(A \underset{\mathbf{HProp}}{=} B\right) \cong (A \cong B)$$

$$\begin{aligned} A \cong B &= \Sigma((f, g) : A \leftrightarrow B, \mathrm{areIso}(f, g)) \\ A \leftrightarrow B &= (A \to B) \times (B \to A) \\ \mathrm{areIso}(f, g) &= \mathbf{Id}(A \to A, g \circ f, \mathrm{id}) \times \mathbf{Id}(B \to B, f \circ g, \mathrm{id}) \end{aligned}$$

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{HProp}, A, B), \cong) \mid \mathrm{idtoiso}\} \cong \{\star\}$$

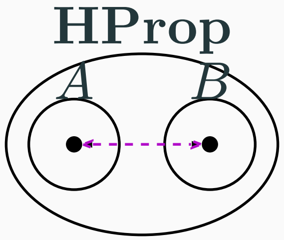
$$\left(A \underset{\mathbf{HProp}}{=} B\right) \cong (A \cong B)$$

$$A \cong B = \Sigma((f, g) : A \leftrightarrow B, \mathrm{areIso}(f, g))$$

$$A \leftrightarrow B = (A \to B) \times (B \to A)$$

$$\mathrm{areIso}(f, g) = \mathbf{Id}(A \to A, g \circ f, \mathrm{id}) \times \mathbf{Id}(B \to B, f \circ g, \mathrm{id})$$

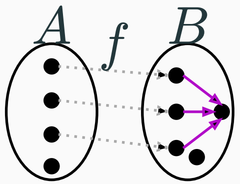
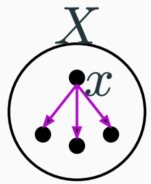
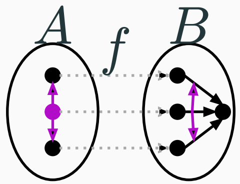
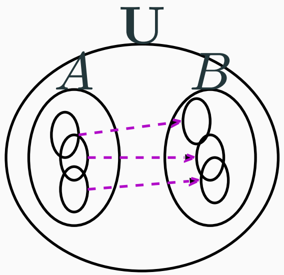
think both types need to have the same amount of terms; this is too restrictive



idtoiso as $p \mapsto (\mathrm{subst}\,\mathrm{id}\,p, \mathrm{subst}\,\mathrm{id}(\mathrm{sym}\ p))$ *and intro an identification given an iso*

5.3 Full Univalence

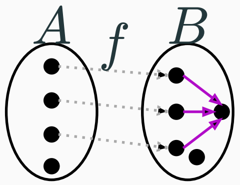
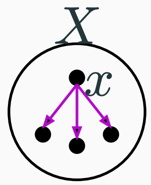
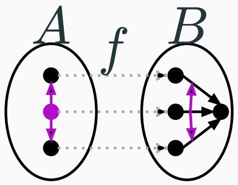
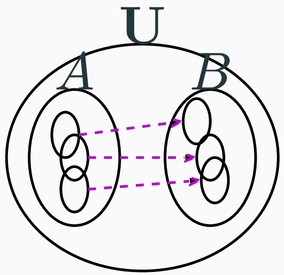
$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{U}, A, B), \simeq) \mid \mathrm{ua}\} \cong \{\star\}$$

fibre	contractible	contractible fibre	univalence
			

5.3 Full Univalence

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{U}, A, B), \simeq) \mid \mathrm{ua}\} \cong \{\star\}$$

$$(A = B) \simeq (A \simeq B)$$

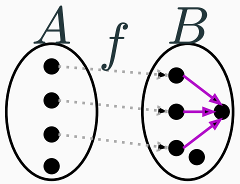
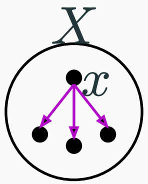
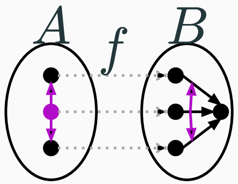
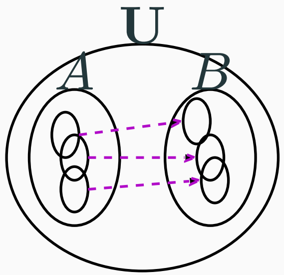
fibre	contractible	contractible fibre	univalence
			

5.3 Full Univalence

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{U}, A, B), \simeq) \mid \mathrm{ua}\} \cong \{\star\}$$

$$(A = B) \simeq (A \simeq B)$$

$$\mathrm{fib}(f, b) = \Sigma(a : A, \mathbf{Id}(B, f\ a, b))$$

fibre	contractible	contractible fibre	univalence
			

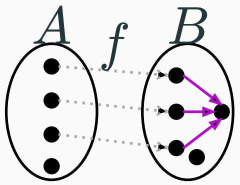
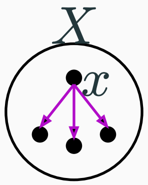
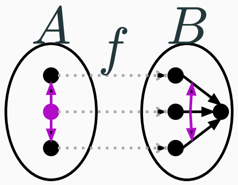
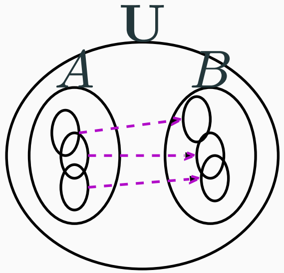
5.3 Full Univalence

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{U}, A, B), \simeq) \mid \text{ua}\} \cong \{\star\}$$

$$(A = B) \simeq (A \simeq B)$$

$$\text{fib}(f, b) = \Sigma(a : A, \mathbf{Id}(B, f\ a, b))$$

$$\text{isContr}(X) = \Sigma(x : X, \Pi(y : X, \mathbf{Id}(X, x, y)))$$

fibre	contractible	contractible fibre	univalence
			

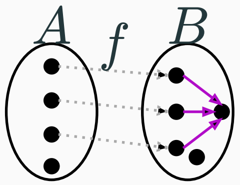
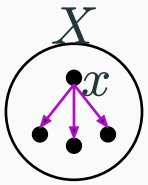
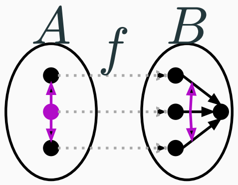
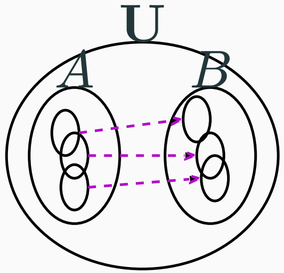
$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{U}, A, B), \simeq) \mid \mathrm{ua}\} \cong \{\star\}$$

$$(A = B) \simeq (A \simeq B)$$

$$\mathrm{fib}(f, b) = \Sigma(a : A, \mathbf{Id}(B, f\ a, b))$$

$$\mathrm{isContr}(X) = \Sigma(x : X, \Pi(y : X, \mathbf{Id}(X, x, y)))$$

$$\mathrm{isEquiv}(f) = \Pi(b : B, \mathrm{isContr}(\mathrm{fib}(f, b)))$$

fibre	contractible	contractible fibre	univalence
			

5.3 Full Univalence

$$\{p \in \mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{U}, A, B), \simeq) \mid \mathrm{ua}\} \cong \{\star\}$$

$$(A = B) \simeq (A \simeq B)$$

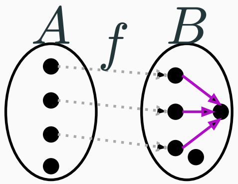
$$\mathrm{fib}(f, b) = \Sigma(a : A, \mathbf{Id}(B, f\ a, b))$$

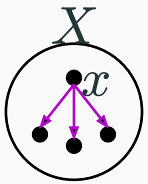
$$\mathrm{isContr}(X) = \Sigma(x : X, \Pi(y : X, \mathbf{Id}(X, x, y)))$$

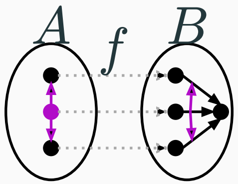
$$\mathrm{isEquiv}(f) = \Pi(b : B, \mathrm{isContr}(\mathrm{fib}(f, b)))$$

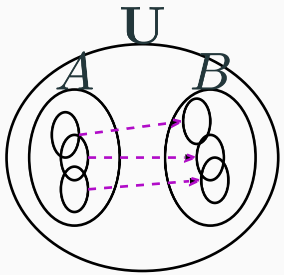
$$A \simeq B = \Sigma(f : A \to B, \mathrm{isEquiv}(f))$$

fibre	contractible	contractible fibre	univalence









6. Conclusion

6.1 Example

- Let N be the type of binary encoded big integers and $\mathbb{N}$ be the peano inductive type of natural numbers

- Let N be the type of binary encoded big integers and $\mathbb{N}$ be the peano inductive type of natural numbers
- lets say we have a function $\text{isEven} : \mathbb{N} \to \mathbf{Bool}$
- given that $p : \mathbf{Id}(\mathbf{U}, N, \mathbb{N})$

6.1 Example

- Let N be the type of binary encoded big integers and $\mathbb{N}$ be the peano inductive type of natural numbers
- lets say we have a function $\text{isEven} : \mathbb{N} \to \mathbf{Bool}$
- given that $p : \mathbf{Id}(\mathbf{U}, N, \mathbb{N})$

$$\text{isEven}' : N \to \mathbf{Bool}$$
$$\text{isEven}'(n) = \text{isEven} \circ (\mathbf{fst} \circ \text{ua } p)\ n$$

- without "using" the structure of N we get the same function on N!
- this is called proof transfer

6.1 Example

- Let N be the type of binary encoded big integers and $\mathbb{N}$ be the peano inductive type of natural numbers
- lets say we have a function isEven : $\mathbb{N} \to \mathbf{Bool}$
- given that $p : \mathbf{Id}(\mathbf{U}, N, \mathbb{N})$

$$\text{isEven}' : N \to \mathbf{Bool}$$
$$\text{isEven}'(n) = \text{isEven} \circ (\mathbf{fst} \circ \text{ua } p)\ n$$

- without "using" the structure of N we get the same function on N!
- this is called proof transfer
- however ua is an axiom without a computational value, thus we can't implement it in a theorem prover
- an attempt at resolving this is using ideas from logical relations and parametricity to get univalent parametricity

6.2 Summary

$$\Gamma \vdash a = b : A$$

- definitional equality by syntax wont allow us to specify equalities depending on a type

$$\Gamma \vdash a = b : A$$

- definitional equality by syntax wont allow us to specify equalities depending on a type

$$\mathbf{Tm}(\Gamma, \mathbf{Eq}(A, a, b))$$

- propositional equality via reflection breaks normalization

$$\Gamma \vdash a = b : A$$

- definitional equality by syntax wont allow us to specify equalities depending on a type

$$\mathbf{Tm}(\Gamma, \mathbf{Eq}(A, a, b))$$

- propositional equality via reflection breaks normalization

$$\mathbf{Tm}(\Gamma.\ \mathbf{Id}(A, a, b), C)$$

- propositional equality via intensional equality hints at higher structures without UIP

$$\Gamma \vdash a = b : A$$

- definitional equality by syntax wont allow us to specify equalities depending on a type

$$\mathbf{Tm}(\Gamma, \mathbf{Eq}(A, a, b))$$

- propositional equality via reflection breaks normalization

$$\mathbf{Tm}(\Gamma.\ \mathbf{Id}(A, a, b), C)$$

- propositional equality via intensional equality hints at higher structures without UIP

$$\mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{U}, A, B), \simeq)$$

- univalence allows us to do proof transfer

$$\Gamma \vdash a = b : A$$

- definitional equality by syntax wont allow us to specify equalities depending on a type

$$\mathbf{Tm}(\Gamma, \mathbf{Eq}(A, a, b))$$

- propositional equality via reflection breaks normalization

$$\mathbf{Tm}(\Gamma.\ \mathbf{Id}(A, a, b), C)$$

- propositional equality via intensional equality hints at higher structures without UIP

$$\mathbf{Tm}(\Gamma.\ \mathbf{Id}(\mathbf{U}, A, B), \simeq)$$

- univalence allows us to do proof transfer ; theoretically, thus we need more