

Martin-Löf's Type Theory

Bas van Gijzel

October 6, 2010

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Outline

Type systems: revisited

Curry-Howard correspondence

Martin-Löf's type theory

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Outline

Type systems: revisited

Curry-Howard correspondence

Martin-Löf's type theory

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Why type systems?

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Why type systems?

“A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.” (Pierce2002)

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Why type systems?

“A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.” (Pierce2002)

- ▶ Normally about **programs**, but too narrow for us.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Why type systems?

“A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.” (Pierce2002)

- ▶ Normally about **programs**, but too narrow for us.
- ▶ **Tractable**, trade-off between manual **type annotation** and automatic **type inference**.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Why type systems?

“A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.” (Pierce2002)

- ▶ Normally about **programs**, but too narrow for us.
- ▶ **Tractable**, trade-off between manual **type annotation** and automatic **type inference**.
- ▶ **Syntactic**, by classification of terms. **Static**.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Why type systems?

“A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.” (Pierce2002)

- ▶ Normally about **programs**, but too narrow for us.
- ▶ **Tractable**, trade-off between manual **type annotation** and automatic **type inference**.
- ▶ **Syntactic**, by classification of terms. **Static**.
- ▶ **Absence**, only categorical absence of bad behaviour can be proved. (rejecting legal programs)

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Type checking

- ▶ Type checking of terms is done automatically by the compiler.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Type checking

- ▶ Type checking of terms is done automatically by the compiler.

Two options: Type annotations versus type inference.

- ▶ **Annotations** are easy to check, but harder to write.
- ▶ **Inference** is easy for the programmer but hard for the compiler.
- ▶ Tractability of **type inference** for dependent types?

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Types and terms

In a typed programming language we want to be able to construct terms and types.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Types and terms

In a typed programming language we want to be able to construct terms and types.

Furthermore, we want to make a connection between both, using a so called typing relation.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Terms

Terms of an example arithmetic language:

```
t ::= true
     false
     if t then t else t
     0
     succ t
     pred t
     t + t
     iszero t
```

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Terms

Terms of an example arithmetic language:

```
t ::= true
    false
    if t then t else t
    0
    succ t
    pred t
    t + t
    iszero t
```

And we want to be able to use a set of **variables**: $\{x, y, x_1, \dots\}$ representing arbitrary terms.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Types

Types of this example arithmetic language:

$T ::= \text{Bool}$
 Nat

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Typing simple expressions

So now we can already type most of our expressions. For example:

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Typing simple expressions

So now we can already type most of our expressions. For example:

$3 : \text{Nat}$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Typing simple expressions

So now we can already type most of our expressions. For example:

$$3 : \text{Nat}$$
$$(\text{if true then } 4 \text{ else } 5) : \text{Nat}.$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Typing simple expressions

So now we can already type most of our expressions. For example:

$3 : \text{Nat}$

$(\text{if true then } 4 \text{ else } 5) : \text{Nat}.$

succ true will not be correctly typed.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Environments (contexts) (1)

But before we're able to fully construct a typing relation we would need to be able to type expression like these:

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Environments (contexts) (1)

But before we're able to fully construct a typing relation we would need to be able to type expression like these:

$x + 3,$
if x then 3 else 5,
etc.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Environments (contexts) (1)

But before we're able to fully construct a typing relation we would need to be able to type expression like these:

$$x + 3,$$
$$\textit{if } x \textit{ then } 3 \textit{ else } 5,$$

etc.

- ▶ We will need to know the **type of variables**.
- ▶ This type depends on a current context/typing environment.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Environments (contexts) (2)

An environment can be build up by starting with the empty environment and by adding bindings.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Environments (contexts) (2)

An environment can be build up by starting with the empty environment and by adding bindings.

$$\Gamma ::= \epsilon \\ | \Gamma, x : T$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Typing relation

We know have the elements needed to construct a typing relation for a given environment.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Typing relation

We know have the elements needed to construct a typing relation for a given environment.

$$\Gamma \vdash t : T$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Typing relation

We know have the elements needed to construct a typing relation for a given environment.

$$\Gamma \vdash t : T$$

- ▶ This statement can be read as:
in environment Γ , the term t has type T .

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Typing relation

We know have the elements needed to construct a typing relation for a given environment.

$$\Gamma \vdash t : T$$

- ▶ This statement can be read as:
in environment Γ , the term t has type T .
- ▶ The statement “ t has type T (given context Γ)” is a type of judgement.
- ▶ For this simply arithmetic language this is the **only** type of judgement.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Type rules

- ▶ To construct the typing relation we use type rules in Gentzen style natural deduction.
- ▶ Inference rules come in two forms, with and without assumptions.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Type rules

- ▶ To construct the typing relation we use type rules in Gentzen style natural deduction.
- ▶ Inference rules come in two forms, with and without assumptions.

$$\frac{}{\Gamma, x:\text{Nat}, \Gamma' \vdash x:\text{Nat}} \text{ (axiom)}$$

$$\frac{\Gamma \vdash M:\text{Nat} \quad \Gamma \vdash N:\text{Nat}}{\Gamma \vdash M+N:\text{Nat}} \text{ (inference rule)}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Outline

Type systems: revisited

Curry-Howard correspondence

Martin-Löf's type theory

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Lambda calculus and natural deduction(1)

Assuming a simply-typed lambda calculus, some standard type (inference) rules would be:

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Lambda calculus and natural deduction(1)

Assuming a simply-typed lambda calculus, some standard type (inference) rules would be:

$$\frac{}{\Gamma_1, x:\alpha, \Gamma_2 \vdash x:\alpha} \text{ (axiom)} \quad \frac{\Gamma, x:\alpha \vdash t:\beta}{\Gamma \vdash \lambda x. t:\alpha \rightarrow \beta} \text{ (function abstraction)}$$

$$\frac{\Gamma \vdash t:\alpha \rightarrow \beta \quad \Gamma \vdash u:\alpha}{\Gamma \vdash t u:\beta} \text{ (function application)}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Lambda calculus and natural deduction(1)

Assuming a simply-typed lambda calculus, some standard type (inference) rules would be:

$$\frac{}{\Gamma_1, x:\alpha, \Gamma_2 \vdash x:\alpha} \text{ (axiom)} \quad \frac{\Gamma, x:\alpha \vdash t:\beta}{\Gamma \vdash \lambda x. t:\alpha \rightarrow \beta} \text{ (function abstraction)}$$

$$\frac{\Gamma \vdash t:\alpha \rightarrow \beta \quad \Gamma \vdash u:\alpha}{\Gamma \vdash t u:\beta} \text{ (function application)}$$

- ▶ Expanding on Curry, Howard noticed a **syntactic correspondence** between simply-typed lambda calculus and natural deduction.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Lambda calculus and natural deduction(2)

► Lambda calculus:

$$\frac{}{\Gamma_1, x:\alpha, \Gamma_2 \vdash x:\alpha} \text{ (axiom)} \quad \frac{\Gamma, x:\alpha \vdash t:\beta}{\Gamma \vdash \lambda x. t:\alpha \rightarrow \beta} \text{ (function abstraction)}$$

$$\frac{\Gamma \vdash t:\alpha \rightarrow \beta \quad \Gamma \vdash u:\alpha}{\Gamma \vdash t \ u:\beta} \text{ (function application)}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Lambda calculus and natural deduction(2)

▶ Lambda calculus:

$$\frac{}{\Gamma_1, x:\alpha, \Gamma_2 \vdash x:\alpha} \text{ (axiom)} \quad \frac{\Gamma, x:\alpha \vdash t:\beta}{\Gamma \vdash \lambda x. t:\alpha \rightarrow \beta} \text{ (function abstraction)}$$

$$\frac{\Gamma \vdash t:\alpha \rightarrow \beta \quad \Gamma \vdash u:\alpha}{\Gamma \vdash t \ u:\beta} \text{ (function application)}$$

▶ (Intuitionistic implicational) natural deduction:

$$\frac{}{\Gamma_1, \alpha, \Gamma_2 \vdash \alpha} \text{ (axiom)} \quad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \rightarrow \beta} \text{ (} \rightarrow \text{ introduction)}$$

$$\frac{\Gamma \vdash \alpha \rightarrow \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \text{ (} \rightarrow \text{ elimination)}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Curry-Howard correspondence (1)

The Curry-Howard correspondence is a correspondence between **propositions** and **types**, and **proofs** and **programs**.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Curry-Howard correspondence (1)

The Curry-Howard correspondence is a correspondence between **propositions** and **types**, and **proofs** and **programs**.

- ▶ The **type of a function** corresponds to a **logical proposition**.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Curry-Howard correspondence (1)

The Curry-Howard correspondence is a correspondence between **propositions** and **types**, and **proofs** and **programs**.

- ▶ The **type of a function** corresponds to a **logical proposition**.
- ▶ The **program (function)** implementing that type corresponds to a **proof** of that logical proposition.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Curry-Howard correspondence (2) (exercise)

Given:

$$\frac{}{\Gamma_1, \alpha, \Gamma_2 \vdash \alpha} \text{ (axiom) } \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \rightarrow \beta} \text{ (} \rightarrow \text{ introduction)}$$

$$\frac{\Gamma \vdash \alpha \rightarrow \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \text{ (} \rightarrow \text{ elimination)}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Curry-Howard correspondence (2) (exercise)

Given:

$$\frac{}{\Gamma_1, \alpha, \Gamma_2 \vdash \alpha} \text{ (axiom)} \quad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \rightarrow \beta} \text{ (} \rightarrow \text{ introduction)}$$

$$\frac{\Gamma \vdash \alpha \rightarrow \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \text{ (} \rightarrow \text{ elimination)}$$

Try proving $\vdash \alpha \rightarrow (\beta \rightarrow \alpha)$!

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Outline

Type systems: revisited

Curry-Howard correspondence

Martin-Löf's type theory

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Type theory: a programming language

- ▶ Per Martin-Löf developed a formalism for constructive mathematics, the *intuitionistic theory of types*.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Type theory: a programming language

- ▶ Per Martin-Löf developed a formalism for constructive mathematics, the *intuitionistic theory of types*.
- ▶ By C-H, this formalism can also be used for **program construction** and is actually well-suited for it.
- ▶ So both **specifications** and **programs** are easily made in the same formalism.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Type theory: a programming language

- ▶ Per Martin-Löf developed a formalism for constructive mathematics, the *intuitionistic theory of types*.
- ▶ By C-H, this formalism can also be used for **program construction** and is actually well-suited for it.
- ▶ So both **specifications** and **programs** are easily made in the same formalism.
- ▶ Constructing a **program** means **proving** it correct and reversely.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Constructive mathematics

- ▶ Proofs should be **constructive**!
 - Classical logic uses rules such as “excluded middle” (tertium non datur) and *reductio ad absurdum* (RAA).

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf’s type
theory



Constructive mathematics

- ▶ Proofs should be **constructive**!
 - Classical logic uses rules such as “excluded middle” (tertium non datur) and *reductio ad absurdum* (RAA).
- ▶ However, we’re not able to prove $\Gamma \vdash \alpha \vee \neg\alpha$ from nothing. This would amount to having a proof, and thus a program, of either one of these propositions.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf’s type
theory



Constructive mathematics

- ▶ Proofs should be **constructive**!
 - Classical logic uses rules such as “excluded middle” (tertium non datur) and *reductio ad absurdum* (RAA).
- ▶ However, we’re not able to prove $\Gamma \vdash \alpha \vee \neg\alpha$ from nothing. This would amount to having a proof, and thus a program, of either one of these propositions.
- ▶ We would need either a proof for α or a proof for $\neg\alpha$ and then construct the disjunction from that.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf’s type
theory



Universiteit Utrecht

Constructive mathematics

- ▶ Proofs should be **constructive**!
 - Classical logic uses rules such as “excluded middle” (tertium non datur) and *reductio ad absurdum* (RAA).
- ▶ However, we’re not able to prove $\Gamma \vdash \alpha \vee \neg\alpha$ from nothing. This would amount to having a proof, and thus a program, of either one of these propositions.
- ▶ We would need either a proof for α or a proof for $\neg\alpha$ and then construct the disjunction from that.
- ▶ Also important, because we want total correctness, the programming language will also need to be **total**.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf’s type
theory



Universiteit Utrecht

Haskell and the Curry-Howard correspondence

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory

Question:

Why do we need totality? (Think of Haskell or fix.)



Formalising constructive mathematics

One of the main issues of formalising constructive mathematics was to fully make the mathematics **explicit**.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Formalising constructive mathematics

One of the main issues of formalising constructive mathematics was to fully make the mathematics **explicit**. This is in contrast to normal natural deduction rules:

$$\frac{A}{A \vee B}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Formalising constructive mathematics

One of the main issues of formalising constructive mathematics was to fully make the mathematics **explicit**. This is in contrast to normal natural deduction rules:

$$\frac{A}{A \vee B}$$

Here it is implicit that A and B are formulas and how they are constructed.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Formalising constructive mathematics

One of the main issues of formalising constructive mathematics was to fully make the mathematics **explicit**. This is in contrast to normal natural deduction rules:

$$\frac{A}{A \vee B}$$

Here it is implicit that A and B are formulas and how they are constructed.

- ▶ So instead:

$$\frac{A \text{ prop.} \quad B \text{ prop.} \quad A \text{ true}}{A \vee B \text{ true}}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Proposition as sets

- ▶ In Martin-Löf's type theory **propositions** are interpreted as sets whose **elements** represent the **proofs of the proposition**.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Proposition as sets

- ▶ In Martin-Löf's type theory **propositions** are interpreted as sets whose **elements** represent the **proofs of the proposition**.
- ▶ The following correspondences hold:

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Proposition as sets

- ▶ In Martin-Löf's type theory **propositions** are interpreted as sets whose **elements** represent the **proofs of the proposition**.
- ▶ The following correspondences hold:
 - $a \in A$.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Proposition as sets

- ▶ In Martin-Löf's type theory **propositions** are interpreted as sets whose **elements** represent the **proofs of the proposition**.
- ▶ The following correspondences hold:
 - $a \in A$.
 - a is a **proof** of the **proposition** A .

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Proposition as sets

- ▶ In Martin-Löf's type theory **propositions** are interpreted as sets whose **elements** represent the **proofs of the proposition**.
- ▶ The following correspondences hold:
 - $a \in A$.
 - a is a **proof** of the **proposition** A .
 - a is an **object** in the **type** A .

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Proposition as sets

- ▶ In Martin-Löf's type theory **propositions** are interpreted as sets whose **elements** represent the **proofs of the proposition**.
- ▶ The following correspondences hold:
 - $a \in A$.
 - a is a **proof** of the **proposition** A .
 - a is an **object** in the **type** A .
 - a is a **program** with the **specification** A .

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Proposition as sets

- ▶ In Martin-Löf's type theory **propositions** are interpreted as sets whose **elements** represent the **proofs of the proposition**.
- ▶ The following correspondences hold:
 - $a \in A$.
 - a is a **proof** of the **proposition** A .
 - a is an **object** in the **type** A .
 - a is a **program** with the **specification** A .
 - a is a **solution** to the **problem** A .

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Judgements

Four (or five) types of judgements:

1. A is a set. (proposition)

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Judgements

Four (or five) types of judgements:

1. A is a set. (proposition)
2. A_1 and A_2 are equal sets. (equal propositions)

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Judgements

Four (or five) types of judgements:

1. A is a set. (proposition)
2. A_1 and A_2 are equal sets. (equal propositions)
3. α is an element in the set A . (α is a proof for proposition A)

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Judgements

Four (or five) types of judgements:

1. A is a set. (proposition)
2. A_1 and A_2 are equal sets. (equal propositions)
3. α is an element in the set A . (α is a proof for proposition A)
4. α_1 and α_2 are equal elements in the set A . (α_1 and α_2 equal proofs for proposition A .)

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Judgements

Four (or five) types of judgements:

1. A is a set. (proposition)
2. A_1 and A_2 are equal sets. (equal propositions)
3. α is an element in the set A . (α is a proof for proposition A)
4. α_1 and α_2 are equal elements in the set A . (α_1 and α_2 equal proofs for proposition A .)
5. A is non-empty. (proposition A is true. This can be seen as 3. without naming a)

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Sets, constructively (1)

Continuing to make concepts explicit, we need to make the semantics of our judgements explicit.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Sets, constructively (1)

Continuing to make concepts explicit, we need to make the semantics of our judgements explicit.

- ▶ What does the judgement “ A is a set” mean?
- ▶ What is a set?

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Sets, constructively (1)

Continuing to make concepts explicit, we need to make the semantics of our judgements explicit.

- ▶ What does the judgement “ A is a set” mean?
- ▶ What is a set?

Characterise its members:

$$\frac{}{0 \in \mathbb{N}} \quad \frac{a \in \mathbb{N}}{a' \in \mathbb{N}}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Sets, constructively (2)

Characterise its members:

$$\frac{}{0 \in \mathbb{N}} \quad \frac{a \in \mathbb{N}}{a' \in \mathbb{N}}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Sets, constructively (2)

Characterise its members:

$$\frac{}{0 \in \mathbb{N}} \quad \frac{a \in \mathbb{N}}{a' \in \mathbb{N}}$$

- ▶ What about 20^{10} ?

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Sets, constructively (2)

Characterise its members:

$$\frac{}{0 \in \mathbb{N}} \quad \frac{a \in \mathbb{N}}{a' \in \mathbb{N}}$$

- ▶ What about 20^{10} ?
- ▶ We need to know when two elements are equal.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Canonical expressions

To be able to define equality we first need the notion of **canonical expression**.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Canonical expressions

To be able to define equality we first need the notion of **canonical expression**. This notion is equivalent to the weak-head normal form in a lazy language such as Haskell.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Canonical expressions

To be able to define equality we first need the notion of **canonical expression**. This notion is equivalent to the weak-head normal form in a lazy language such as Haskell.

- ▶ For equality on expressions, we need equality of canonical expressions.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Canonical expressions

To be able to define equality we first need the notion of **canonical expression**. This notion is equivalent to the weak-head normal form in a lazy language such as Haskell.

- ▶ For equality on expressions, we need equality of canonical expressions.
- ▶ Similar to before, we need this to be **explicit**.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Canonical expressions

To be able to define equality we first need the notion of **canonical expression**. This notion is equivalent to the weak-head normal form in a lazy language such as Haskell.

- ▶ For equality on expressions, we need equality of canonical expressions.
- ▶ Similar to before, we need this to be **explicit**.
- ▶ This is reflected in another type of judgement, “ α_1 and α_2 are equal elements in the set A ”.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Canonical expressions

To be able to define equality we first need the notion of **canonical expression**. This notion is equivalent to the weak-head normal form in a lazy language such as Haskell.

- ▶ For equality on expressions, we need equality of canonical expressions.
- ▶ Similar to before, we need this to be **explicit**.
- ▶ This is reflected in another type of judgement, “ α_1 and α_2 are equal elements in the set A ”.
- ▶ Equality between canonical expressions should be reflexive, transitive and symmetric.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Definition of a set

The **definition** of a set (Nordström(1990)):

“To know that A is a set is to know how to form the canonical elements in the set and under what conditions two canonical elements are equal.”

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Definition of a set

The **definition** of a set (Nordström(1990)):

“To know that A is a set is to know how to form the canonical elements in the set and under what conditions two canonical elements are equal.”

To **construct** a set (Nordström(1990)):

- ▶ *“Give a prescription of how to form (construct) the canonical elements, i.e. define the syntax of the canonical expressions and the premises for forming them.”*
- ▶ *“Give the premises for forming two equal canonical elements.”*

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht

Example of a set

Again:

$$\overline{0 \in \mathbb{N}} \quad \frac{a \in \mathbb{N}}{a' \in \mathbb{N}}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Example of a set

Again:

$$\frac{}{0 \in \mathbb{N}} \quad \frac{a \in \mathbb{N}}{a' \in \mathbb{N}}$$

And now also:

$$\frac{}{0 = 0} \quad \frac{a = b \in \mathbb{N}}{a' = b' \in \mathbb{N}}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



General rules of equality

Some rules for equality:

$$\frac{a \in A}{a = a \in A} \quad \frac{A \text{ type}}{A = A} \text{ (Reflexivity)}$$

$$\frac{a = b \in A \quad b = c \in A}{a = c \in A} \quad \frac{A = B \quad B = C}{A = C} \text{ (Transitivity)}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Definitions of other judgements etc.

Similar to the concepts of sets we need to define semantics of judgements under assumptions, etc.

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Definitions of other judgements etc.

Similar to the concepts of sets we need to define semantics of judgements under assumptions, etc.

(See Nordström(1990) or Per Martin-Löf (1984b).)

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Martin L of and dependent types

- ▶ Similar to the general rules for equality and union, Martin L of's type theory also has constructors for products.

Type systems:
revisited

Curry-Howard
correspondence

Martin-L of's type
theory



Martin L of and dependent types

- ▶ Similar to the general rules for equality and union, Martin L of's type theory also has constructors for products.
- ▶ The argument of the products can be dependent however.

Type systems:
revisited

Curry-Howard
correspondence

Martin-L of's type
theory



Martin L of and dependent types

- ▶ Similar to the general rules for equality and union, Martin L of's type theory also has constructors for products.
- ▶ The argument of the products can be dependent however.

$$\frac{A \text{ type} \quad (x \in A) \quad B(x) \text{ type}}{\Pi(A,B) \text{ type}}$$

Type systems:
revisited

Curry-Howard
correspondence

Martin-L of's type
theory



Martin L of and dependent types

- ▶ Similar to the general rules for equality and union, Martin L of's type theory also has constructors for products.
- ▶ The argument of the products can be dependent however.

$$\frac{(x \in A) \quad A \text{ type} \quad B(x) \text{ type}}{\Pi(A,B) \text{ type}}$$

For example take for $A: \mathbb{N}$, and for $B(n): \text{BoolVectors}$ taking a natural number as length argument.

Type systems:
revisited

Curry-Howard
correspondence

Martin-L of's type
theory



Agda and the Curry-Howard correspondence

- ▶ *“Agda is essentially an extension of intensional Martin-Löf type theory based on a logical framework.”*
(Agda wiki)
- ▶ Agda is total and we can easily see the correspondence with Martin-Löf.

`data \top : Set where` (True proposition)

`tt : \top`

`data \perp : Set where` (False proposition)

`data $_ \uplus _$ (A B : Set) where` (Disjunction)

`inj1 : A \rightarrow A \uplus B`

`inj2 : B \rightarrow A \uplus B`

Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Questions



Type systems:
revisited

Curry-Howard
correspondence

Martin-Löf's type
theory



Universiteit Utrecht