A Short Course on Program Extraction

Monika Seisenberger

Swansea University

Verona, 26-29 May 2020
Aims of the course

Main question of the course:

*What is the computational content behind a proof?*
Aims of the course

Main question of the course:

What is the computational content behind a proof?

To this end, we will demonstrate that logic is a natural bridge between mathematics and computation.
Aims of the course

Main question of the course:

*What is the computational content behind a proof?*

To this end, we will demonstrate that *logic* is a natural bridge between *mathematics* and *computation*.

We will study how valid reasoning in abstract mathematics leads to provably correct algorithms and hence *certified computer programs*. 
What do we need for this endeavor?

1) A proof calculus (or in general the ability to express statements in logic and to do proofs).
What do we need for this endeavor?

1) A proof calculus (or in general the ability to express statements in logic and to do proofs).

2) A tool that can carry out the proofs for us.
What do we need for this endeavor?

1) A proof calculus (or in general the ability to express statements in logic and to do proofs).

2) A tool that can carry out the proofs for us.

3) A way to store the proofs.
What do we need for this endeavor?

1) A proof calculus (or in general the ability to express statements in logic and to do proofs).

2) A tool that can carry out the proofs for us.

3) A way to store the proofs.

4) A mechanism to extract the computational content from a proof.
What do we need for this endeavor? (2)

1) A proof calculus (or in general the ability to express statements in logic and to do proofs). [Natural deduction]

2) A tool that can carry out the proofs for us. [The Minlog proof system]

3) A way to write up and store the proofs. [\lambda\text{calculus}]

4) A mechanism to extract the computational content from a proof. [Realizability]

5) Interesting case studies and applications.
Outline

Part 1  Logic and Program Extraction
  1.1  Natural Deduction
  1.2  $\lambda$-calculus, Curry Howard-Correspondence
  1.3  Tool support
  1.4  Realizability Interpretation

Part 2  Extensions of the Mechanism to Inductive Definitions and Classical proofs. Applications.
  2.1  Inductive Definitions
  2.2  A-translation, Choice principles
  2.3  Applications for both parts
The fundamental idea of program extraction
The fundamental idea of program extraction

A *proof* is a construction, represented by a text or a finite tree, that convinces us that a formula is *true*. 

The formula stating that there are infinitely many prime numbers,

\[ \forall x \exists y (y > x \land \text{Prime}(y)) \]

can be understood as the problem of computing for every natural number \( x \) a prime number \( y \) that is greater than \( x \). Program extraction is based on the observation that a proof not only represents an argument why a formula is true but also contains a program that solves the computational problem it expresses.
The fundamental idea of program extraction

A *proof* is a construction, represented by a text or a finite tree, that convinces us that a formula is *true*.

Often, a formula can also be understood as a *computational problem*. 
The fundamental idea of program extraction

A \textit{proof} is a construction, represented by a text or a finite tree, that convinces us that a formula is \textit{true}.

Often, a formula can also be understood as a \textit{computational problem}.

For example, the formula stating that there are infinitely many prime numbers,

$$\forall x \exists y \ (y > x \land \text{Prime}(y))$$

can be understood as the problem of computing for every natural number $x$ a prime number $y$ that is greater than $x$. 
The fundamental idea of program extraction

A *proof* is a construction, represented by a text or a finite tree, that convinces us that a formula is *true*.

Often, a formula can also be understood as a *computational problem*.

For example, the formula stating that there are infinitely many prime numbers,

\[ \forall x \exists y \ (y > x \land \text{Prime}(y)) \]

can be understood as the problem of computing for every natural number \( x \) a prime number \( y \) that is greater than \( x \).

**Program extraction** is based on the observation that a proof not only represents an argument why a formula is true but also contains a program that solves the computational problem it expresses.
Predicate logic (a.k.a. first-order logic, FOL)

Gottlob Frege (1848 - 1925)

Predicate logic was introduced by Frege in his *Begriffsschrift*.
The language of predicate logic

Example: "Every positive number has a positive square root"

∀x (x > 0 → ∃y (y > 0 ∧ x = y ∗ y))

The language, \( L = (C, F, P) \), for this formula consists of:
- **Constants**: \( C = \{0\} \)
- **Function symbols**: \( F = \{∗\} \)
- **Predicate symbols**: \( P = \{>\} \)

The elements of \( L \) are also called **non-logical symbols**. The choice of \( L \) may vary depending on the intended application.

The other symbols occurring in a formula of predicate logic are application independent and are called **logical symbols**:
- **Variables**: \( x, y, \ldots \)
- **Logical constants**: \( \top \) ("true"), \( \bot \) ("false")
- **Logical connectives**: \( \land \) ("and"), \( \lor \) ("or"), \( \rightarrow \) ("implies")
- **Quantifiers**: \( \forall \) ("for all"), \( \exists \) ("exists")

Equality: =

Negation can be defined as \( \neg A \) Def = \( A → \bot \).
The language of predicate logic

Example: “Every positive number has a positive square root”
The language of predicate logic

Example: “Every positive number has a positive square root”

\[ \forall x (x > 0 \rightarrow \exists y (y > 0 \land x = y \times y)) \]
The language of predicate logic

Example: “Every positive number has a positive square root”

\[ \forall x \ (x > 0 \rightarrow \exists y \ (y > 0 \land x = y \times y)) \]

The language, \( \mathcal{L} = (\mathcal{C}, \mathcal{F}, \mathcal{P}) \), for this formula consists of

- **Constants**: \( \mathcal{C} = \{0\} \)
- **Function symbols**: \( \mathcal{F} = \{\ast\} \)
- **Predicate symbols**: \( \mathcal{P} = \{>\} \)
The language of predicate logic

Example: “Every positive number has a positive square root”

\[ \forall x \ (x > 0 \rightarrow \exists y \ (y > 0 \land x = y \times y)) \]

The language, \( L = (C, F, P) \), for this formula consists of

- **Constants**: \( C = \{0\} \)
- **Function symbols**: \( F = \{\ast\} \)
- **Predicate symbols**: \( P = \{>\} \)

The elements of \( L \) are also called *non-logical symbols*. The choice of \( L \) may vary depending on the intended application.
The language of predicate logic

Example: “Every positive number has a positive square root”

$$\forall x (x > 0 \rightarrow \exists y (y > 0 \land x = y \ast y))$$

The language, $$\mathcal{L} = (\mathcal{C}, \mathcal{F}, \mathcal{P})$$, for this formula consists of:

- **Constants**: $$\mathcal{C} = \{0\}$$
- **Function symbols**: $$\mathcal{F} = \{\ast\}$$
- **Predicate symbols**: $$\mathcal{P} = \{>\}$$

The elements of $$\mathcal{L}$$ are also called **non-logical symbols**. The choice of $$\mathcal{L}$$ may vary depending on the intended application.

The other symbols occurring in a formula of predicate logic are application independent and are called **logical symbols**:

- **Variables**: $$x, y, \ldots$$
- **Logical constants**: $$\top$$ (“true”), ‘⊥’ (false)
- **Logical connectives**: $$\land$$ (“and”), $$\lor$$ (“or”), $$\rightarrow$$ (“implies”)
- **Quantifiers**: $$\forall$$ (“for all”), $$\exists$$ (“exists”)
- **Equality**: $$=$$

Negation can be defined as $$\neg A \overset{\text{Def}}{=} A \rightarrow \bot.$$
The semantics of predicate logic

Alfred Tarski (1901-1983)

Tarski was the first to systematically study the notion of truth for formulas in predicate logic.
A *model* (or *structure*) $\mathcal{M}$ for a language $\mathcal{L} = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ consists of:

- a nonempty set $M$, called the *carrier set of $\mathcal{M}$*
- an interpretation in $M$ of
  - the constants in $\mathcal{C}$,
  - the function symbols in $\mathcal{F}$,
  - the predicate symbols in $\mathcal{P}$.

In a given model $\mathcal{M}$, any $\mathcal{L}$-formula is either true or false.
Truth, Validity, Logical Consequence

\[ \mathcal{M} \models A \quad (\text{formula } A \text{ is true in model } \mathcal{M}, \text{ or } \mathcal{M} \text{ satisfies } A) \]

A is logically valid \((\models A) \quad \text{Def} \quad \text{for all models } \mathcal{M}, \mathcal{M} \models A. \)

(A is true in all models)

A is a logical consequence of a set of formulas \( \Gamma \) \((\Gamma \models A) \quad \text{Def} \quad \text{for all models } \mathcal{M}, \text{ if } \mathcal{M} \models \Gamma, \text{ then } \mathcal{M} \models A. \)

(A is true in all models of \( \Gamma \), or \( \Gamma \) logically implies \( A \))

Where \( \mathcal{M} \models \Gamma \) means \( \mathcal{M} \models B \) for all \( B \in \Gamma \).
Proofs

A *proof system* is a collection of rules to derive logically valid formulas.

There are many different proof systems. A popular, due to Gentzen, is called *Natural Deduction* since its rules are close to natural human reasoning.

Gerhard Gentzen (1909 - 1945)
# Natural Deduction

<table>
<thead>
<tr>
<th>Assumption rule</th>
<th>Introduction rules</th>
<th>Elimination rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \land )</td>
<td>( \frac{A}{A \land B} \quad \frac{B}{A \land B} \quad \land^+ )</td>
<td>( \frac{A \land B}{A} \quad \land_l^- ) ( \frac{A \land B}{B} \quad \land_r^- )</td>
</tr>
<tr>
<td>( \to )</td>
<td>( \frac{B}{A \to B} \quad \to^+ u : A )</td>
<td>( \frac{A \to B}{A} \quad \to^- ) ( \frac{B}{A} \quad \to^- )</td>
</tr>
<tr>
<td>( \lor )</td>
<td>( \frac{A}{A \lor B} \quad \lor^+ ) ( \frac{B}{A \lor B} \quad \lor_r^+ )</td>
<td>( \frac{A \lor B}{A} \quad A \to C \quad B \to C \quad \lor^- )</td>
</tr>
<tr>
<td>( \bot )</td>
<td>( \frac{\bot}{A} \quad \mathsf{efq} )</td>
<td>( \frac{\neg \neg A}{A} \quad \mathsf{raa} )</td>
</tr>
</tbody>
</table>
### Quantifier rules

<table>
<thead>
<tr>
<th></th>
<th>Introduction rules</th>
<th>Elimination rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall$</td>
<td>$\frac{A(x)}{\forall x \ A(x)}$  ($\forall^+$)</td>
<td>$\frac{\forall x \ A(x)}{A(t)}$  ($\forall^-$)</td>
</tr>
<tr>
<td>$\exists$</td>
<td>$\frac{A(t)}{\exists x \ A(x)}$  ($\exists^+$)</td>
<td>$\frac{\exists x \ A(x)}{\forall x (A(x) \rightarrow C)}$  ($\exists^-$)  (***)</td>
</tr>
</tbody>
</table>

**Variable conditions:**

(*) $x$ must not occur free in any free (that is, uncancelled) assumption.

(**) $x$ must not occur free in $C$.

Adding these rules to natural deduction yields a complete proof system.
# Natural Deduction (version with explicit assumptions)

<table>
<thead>
<tr>
<th>Assumption rule</th>
<th>use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma, A \vdash A$</td>
<td></td>
</tr>
</tbody>
</table>

## Introduction rules

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Introduction rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\land$</td>
<td>$\frac{\Gamma \vdash A}{\Gamma \vdash A \land B}$ $\land^+$</td>
</tr>
<tr>
<td>$\rightarrow$</td>
<td>$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}$ $\rightarrow^+$</td>
</tr>
<tr>
<td>$\lor$</td>
<td>$\frac{\Gamma \vdash A}{\Gamma \vdash A \lor B}$ $\lor^+$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\frac{\Gamma \vdash \bot}{\Gamma \vdash \bot}$ efq</td>
</tr>
</tbody>
</table>

## Elimination rules

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Elimination rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\land$</td>
<td>$\frac{\Gamma \vdash A \land B}{\Gamma \vdash A}$ $\land^-$ $\frac{\Gamma \vdash A \land B}{\Gamma \vdash B}$ $\land^-_r$</td>
</tr>
<tr>
<td>$\rightarrow$</td>
<td>$\frac{\Gamma \vdash A \rightarrow B}{\Gamma \vdash B}$ $\rightarrow^-$</td>
</tr>
<tr>
<td>$\lor$</td>
<td>$\frac{\Gamma \vdash A \lor B}{\Gamma \vdash A}$ $\lor^-$ $\frac{\Gamma \vdash A \lor B}{\Gamma \vdash B}$ $\lor^-_r$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\frac{\Gamma \vdash \bot}{\Gamma \vdash \bot}$ efq $\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$ raa</td>
</tr>
</tbody>
</table>

## Quantifiers

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Quantifier rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall$</td>
<td>$\frac{\Gamma \vdash A(x)}{\Gamma \vdash \forall x A(x)}$ $\forall^+$ $(x \text{ not free in } \Gamma)$</td>
</tr>
<tr>
<td>$\exists$</td>
<td>$\frac{\Gamma \vdash \exists x A(x)}{\Gamma \vdash A(t)}$ $\exists^+$</td>
</tr>
</tbody>
</table>

$x$ not free in $\Gamma$, $C$
### Equality rules (both versions)

<table>
<thead>
<tr>
<th>Introduction rule</th>
<th>Elimination rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$= \quad t = t$</td>
<td>$\frac{A(s) \quad s = t}{A(t)}$</td>
</tr>
<tr>
<td>$\Gamma \vdash t = t$</td>
<td>$\Gamma \vdash A(t)$</td>
</tr>
</tbody>
</table>

Symmetry and transitivity of equality can be derived from these rules.
Example: Equivalence of $A \rightarrow (B \rightarrow C)$ and $A \land B \rightarrow C$

\[
\begin{align*}
  u : A \land B \rightarrow C \\
  v : A & \quad w : B \\
  \frac{C}{A \land B} \rightarrow^- \\
  \frac{B \rightarrow C}{A \rightarrow (B \rightarrow C)} \rightarrow^+ w : B \\
  \frac{A \rightarrow (B \rightarrow C)}{(A \land B \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C))} \rightarrow^+ v : A \\
  \end{align*}
\]

Exercise: Prove other direction on your own.
The Brouwer-Heyting-Kolmogorov Interpretation:

According to the BHK interpretation a formula expresses a *computational problem* which is defined by a description of how to solve it:

A solution to $A \land B$ is a pair $(a, b)$ such that

$$a \text{ solves } A \text{ and } b \text{ solves } B.$$
The Brouwer-Heyting-Kolmogorov Interpretation:

According to the BHK interpretation a formula expresses a \textit{computational problem} which is defined by a description of how to solve it:

A solution to $A \land B$ is a pair $(a, b)$ such that

\[ a \text{ solves } A \text{ and } b \text{ solves } B. \]

A solution to $A \lor B$ is

either $(0, a)$ where $a$ solves $A$

or $(1, b)$ where $b$ solves $B$.

A solution to $A \rightarrow B$ is a construction that transforms

any solution of $A$ to a solution of $B$. 
The lambda calculus

In the BHK interpretation it is left open what a “construction” is.

Church’s lambda calculus provides a good notion of construction:

The lambda calculus consists of

- lambda terms generated by the rules
  - Variables: $x$
  - Lambda-abstraction: $\lambda x . M$
  - Application: $M N$

- beta-reduction

$$(\lambda x . M) N \rightarrow_\beta M[N/x]$$

$M[N/x]$ denotes substitution of the term $N$ for $x$ in the term $M$.

One usually writes $M N K$ for $(M N) K$. 
Lambda calculus with types

*Types* are like propositional formulas with \( \to \) as the only connective.

Let \( \Gamma = x_1 : A_1, \ldots, x_n : A_n \) be a *context*, that is a type assignment to variables.

We define inductively the relation \( \Gamma \vdash M : A \) (\( M \) has type \( A \) in context \( \Gamma \)).

\[
\begin{align*}
\Gamma, x : A & \vdash x : A \\
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x \ M : A \to B} & \quad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}
\end{align*}
\]
β-reduction and β-equality

Theorem

β-reduction is *strongly normalizing*, that is, every reduction sequence

\[ M_1 \rightarrow_\beta M_2 \rightarrow_\beta M_3, \ldots \text{ terminates.} \]

Theorem

β-reduction is *confluent*, that is, if \( M \rightarrow^* N_1 \) and \( M \rightarrow^* N_2 \), then there exists a term \( N \) such that \( N_1 \rightarrow^*_\beta N \) and \( N_2 \rightarrow^*_\beta N \).

Theorem

The relation of β-equality, defined by

\[ M =_\beta N \iff \exists K \ (M \rightarrow^*_\beta K \land N \rightarrow^*_\beta K) \]

is decidable.
Extension to products and sums

\[ \Gamma \vdash M : A \quad \Gamma \vdash N : B \]
\[ \Gamma \vdash (M, N) : A \times B \]

\[ \Gamma \vdash M : A \times B \]
\[ \Gamma \vdash \pi_0(M) : A \]
\[ \Gamma \vdash \pi_1(M) : B \]

\[ \Gamma \vdash M : A \]
\[ \Gamma \vdash (0, M) : A + B \]
\[ \Gamma \vdash (1, M) : A + B \]

\[ \Gamma \vdash M : A + B \quad \Gamma \vdash N : A \to C \quad \Gamma \vdash K : B \to C \]
\[ \Gamma \vdash \text{case}(M, N, K) \]

\[ \pi_0(M, N) \to_\beta M \]
\[ \pi_1(M, N) \to_\beta N \]
\[ \text{case}((0, M), N, K) \to_\beta N M \]
\[ \text{case}((1, M), N, K) \to_\beta K M \]
The Curry-Howard correspondence

The *Curry-Howard correspondence* is the observation that intuitionistic natural deduction proofs are in a natural correspondence with the typed lambda calculus.

Since typed lambda terms are the core of functional programming languages such as ML and Haskell (named after Haskell B Curry) one can also say that intuitionistic proofs correspond to programs.

Haskell B Curry (1900-1982)
### Intuitionistic ND proofs vs typed lambda lambda calculus

<table>
<thead>
<tr>
<th>Intuitionistic logic</th>
<th>Typed lambda calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \land B )</td>
<td>( M : A \quad N : B )</td>
</tr>
<tr>
<td>( A \quad B )</td>
<td>( (M, N) : A \times B )</td>
</tr>
<tr>
<td>( A \land B )</td>
<td>( M : A \times B )</td>
</tr>
<tr>
<td>( A \quad B )</td>
<td></td>
</tr>
<tr>
<td>( A \quad B )</td>
<td>( \pi_0(M) : A )</td>
</tr>
<tr>
<td>( A \quad B )</td>
<td>( \pi_1(M) : B )</td>
</tr>
<tr>
<td>( B \quad \rightarrow^+ u : A )</td>
<td>( M : B )</td>
</tr>
<tr>
<td>( A \rightarrow B )</td>
<td>( \lambda x M : A \rightarrow B )</td>
</tr>
<tr>
<td>( A \rightarrow B )</td>
<td>( M : A \rightarrow B )</td>
</tr>
<tr>
<td>( A )</td>
<td>( N : B )</td>
</tr>
<tr>
<td>( A )</td>
<td>( M N : B )</td>
</tr>
<tr>
<td>( B )</td>
<td></td>
</tr>
<tr>
<td>( A \lor B )</td>
<td>( M : A \lor B )</td>
</tr>
<tr>
<td>( B )</td>
<td></td>
</tr>
<tr>
<td>( A \lor B )</td>
<td>( (0, M) : A \lor B )</td>
</tr>
<tr>
<td>( A \lor B )</td>
<td>( (1, M) : A \lor B )</td>
</tr>
<tr>
<td>( A \rightarrow C \quad B \rightarrow C )</td>
<td>( M : A \lor B )</td>
</tr>
<tr>
<td>( B \rightarrow C )</td>
<td></td>
</tr>
<tr>
<td>( A \lor B )</td>
<td>( N : A \rightarrow C )</td>
</tr>
<tr>
<td>( B \rightarrow C )</td>
<td>( K : B \rightarrow C )</td>
</tr>
<tr>
<td>( A \lor B )</td>
<td>( \text{case}(M, N, K) : C )</td>
</tr>
</tbody>
</table>