Computing in terms of constructive logics

Paweł Urzyczyn
urzy@mimuw.edu.pl

November 10, 2016

General idea

Various aspects of computing can be explained using constructive logics: intuitionistic logic,

General idea

Various aspects of computing can be explained using constructive logics: intuitionistic logic, linear logic,

General idea

Various aspects of computing can be explained using constructive logics: intuitionistic logic, linear logic, intersection logic,

General idea

Various aspects of computing can be explained using constructive logics: intuitionistic logic, linear logic, intersection logic, classical logic ...

Research directions

- Proof-as-computation: lambda-calculi as proof notation.

Research directions

- Proof-as-computation: lambda-calculi as proof notation.
- Computations-as-proofs: synthesis into proof-search.

Research directions

- Proof-as-computation: lambda-calculi as proof notation.
- Computations-as-proofs: synthesis into proof-search.
- Complexity of various logics and their fragments.

Research directions

- Proof-as-computation: lambda-calculi as proof notation.
- Computations-as-proofs: synthesis into proof-search.
- Complexity of various logics and their fragments.
- Related issues: normalization, proof-search games, etc.

Classical logic:The logical value paradigm

- Formula = a declarative assertion about some "reality".
- Correctness criterion = truth (logical value).
- Semantics has priority over proof.
- Proof = just a tool; its shape is not essential, we only ask if one exists.
- Logic = determining if a sentence is true.
- The role of a logical connective = define the truth of a compound sentence in terms of the values of its components.

Intuitionistic logic: The construction paradigm

- Reasoning is primary, semantics is just a tool.
- Correctness criterion = construction.
- The role of a logical connective = express a construction of a compound sentence in terms of constructions of its components.

BHK Interpretation

- *A construction of $\alpha \wedge \beta$ is a pair of constructions, one for α and one for β.*
- *A construction of $\alpha \vee \beta$ consists of a construction of either α or β (and an indication which one).*

Brouwer-Heyting-Kolmogorov

- *A construction of $\alpha \to \beta$ is a method that turns any possible construction of α into a construction of β.*
- *There is no construction of $\bot$.*

Negation: $\neg\varphi = \varphi \to \bot$.

- *A construction of $\neg\varphi$ is a method that turns any hypothetical construction of φ into a nonsense ($\bot$: "the thing which is not").*

Examples of intuitionistic theorems:

- $p \to q \to p$;
- $(p \to q \to r) \to (p \to q) \to p \to r$;
- $(p \to q) \to (\neg q \to \neg p)$;
- $p \to \neg\neg p$;
- $(p \vee \neg p) \to \neg\neg p \to p$;
- $(p \to \neg q) \to (\neg p \to \neg q) \to \neg q$;
- $\neg\neg(p \vee \neg p)$;
- $\bot \to p$.

Examples of intuitionistic non-theorems:

- $\neg\neg p \to p$;
- $\neg p \vee p$;
- $(\neg q \to \neg p) \to (p \to q)$;
- $\neg(p \wedge q) \to (\neg p \vee \neg q)$;
- $(p \to q) \to (\neg p \to q) \to q$;
- $p \vee (p \to q)$;
- $((p \to q) \to p) \to p$.

Natural deduction

$$\Gamma, \sigma \vdash \sigma$$

$$\frac{\Gamma, \sigma \vdash \tau}{\Gamma \vdash \sigma \to \tau}$$

$$\frac{\Gamma \vdash \sigma \to \tau \qquad \Gamma \vdash \sigma}{\Gamma \vdash \tau}$$

In the meantime. . .

In the meantime. . .

. . . Alonzo Church invented λ-calculus.

The untyped λ-calculus

Terms:

- Variables x, y, z,...
- Applications (MN);
- Abstractions $\lambda x.\, M$.

Beta-reduction;

$$(\lambda x.\, M)N \quad \Rightarrow \quad M[x := N]$$

Simply typed lambda-calculus

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma . M) : \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (MN) : \tau}$$

Back to natural deduction

$$\Gamma, \sigma \vdash \sigma$$

$$\frac{\Gamma, \sigma \vdash \tau}{\Gamma \vdash \sigma \to \tau}$$

$$\frac{\Gamma \vdash \sigma \to \tau \qquad \Gamma \vdash \sigma}{\Gamma \vdash \tau}$$

Proof notation for natural deduction

$$\Gamma, \quad \sigma \vdash \quad \sigma$$

$$\frac{\Gamma, \quad \sigma \vdash \quad \tau}{\Gamma \vdash \qquad\qquad \sigma \to \tau}$$

$$\frac{\Gamma \vdash \quad \sigma \to \tau \qquad\qquad \Gamma \vdash \quad \sigma}{\Gamma \vdash \qquad\quad \tau}$$

Proof notation for natural deduction

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma, \quad \sigma \vdash \quad \tau}{\Gamma \vdash \qquad \sigma \to \tau}$$

$$\frac{\Gamma \vdash \quad \sigma \to \tau \qquad\qquad \Gamma \vdash \quad \sigma}{\Gamma \vdash \qquad \tau}$$

Proof notation for natural deduction

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma, \quad \sigma \vdash \quad \tau}{\Gamma \vdash \qquad \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad\qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \qquad\qquad \tau}$$

Proof notation for natural deduction

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma, \quad \sigma \vdash \quad \tau}{\Gamma \vdash \quad \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M \mathbin{@} N : \tau}$$

Proof notation for natural deduction

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M \mathbin{@} N : \tau}$$

Proof notation for natural deduction

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash M \lightning x : \sigma : \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M \mathbin{@} N : \tau}$$

Proof notation for natural deduction

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lightning x : \sigma\, M : \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M \mathbin{@} N : \tau}$$

Proof notation for natural deduction

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.\, M : \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (M\, N) : \tau}$$

Curry-Howard Isomorphism

An implicational formula α is an intuitionistic theorem iff there exists a closed term of type α (type α is nonempty). This extends to $\vee$, $\wedge$, $\bot$, $\forall$, $\exists$, $\ldots$

Curry-Howard Isomorphism

An implicational formula α is an intuitionistic theorem iff there exists a closed term of type α (type α is nonempty). This extends to $\vee$, $\wedge$, $\bot$, $\forall$, $\exists$, ...

Normalization

$$(\lambda x{:}\sigma.M)N \Rightarrow M[x := N]$$

Curry-Howard Isomorphism

An implicational formula α is an intuitionistic theorem iff there exists a closed term of type α (type α is nonempty). This extends to $\vee$, $\wedge$, $\bot$, $\forall$, $\exists$, ...

Normalization

$$(\lambda x{:}\sigma.M)N \Rightarrow M[x := N]$$

Every reduction sequence leads to a *normal form*.

Curry-Howard Isomorphism

An implicational formula α is an intuitionistic theorem iff there exists a closed term of type α (type α is nonempty). This extends to $\vee$, $\wedge$, $\bot$, $\forall$, $\exists$, ...

Normalization

$$(\lambda x{:}\sigma.M)N \Rightarrow M[x := N]$$

Every reduction sequence leads to a *normal form*.

Consistency: There is no normal term of type $\bot$

Curry, Howard, de Bruijn,...
"Propositions-as-Types"

- Formula = type = specification.
- Proof = program = implementation.
- Proof normalization = computation.

Research directions

- Proof-as-computation: lambda-calculi as proof notation.

Research directions

- Proof-as-computation: lambda-calculi as proof notation.

We study properties of various lambda-calculi representing different logics and different computational paradigms. For example, the polymorphic lambda-calculus, aka "system F", corresponds to second-order propositional intuitionistic logic.

Problems:

- Normalization: does every computation terminate?

Research directions

- Proof-as-computation: lambda-calculi as proof notation.

We study properties of various lambda-calculi representing different logics and different computational paradigms. For example, the polymorphic lambda-calculus, aka "system F", corresponds to second-order propositional intuitionistic logic.

Problems:

- Normalization: does every computation terminate?
- Translatability: how to "embed" one calculus into another?

Research directions

- Proof-as-computation: lambda-calculi as proof notation.

We study properties of various lambda-calculi representing different logics and different computational paradigms. For example, the polymorphic lambda-calculus, aka "system F", corresponds to second-order propositional intuitionistic logic.

Problems:

- Normalization: does every computation terminate?
- Translatability: how to "embed" one calculus into another?
- Inhabitation: does there exist a term of a given type?

Research directions

- Proof-as-computation: lambda-calculi as proof notation.

We study properties of various lambda-calculi representing different logics and different computational paradigms. For example, the polymorphic lambda-calculus, aka "system F", corresponds to second-order propositional intuitionistic logic.

Problems:

- Normalization: does every computation terminate?
- Translatability: how to "embed" one calculus into another?
- Inhabitation: does there exist a term of a given type?
- Expressive power: what can be defined?

Research directions

- Computations-as-proofs: synthesis into proof-search.

Research directions

- Computations-as-proofs: synthesis into proof-search.

A specification of a computational problem, esp. synthesis problem (to define a certain function) can be seen as a formula of some logic. A proof of the formula yields a solution (e.g. a program)

Research directions

- Computations-as-proofs: synthesis into proof-search.

A specification of a computational problem, esp. synthesis problem (to define a certain function) can be seen as a formula of some logic. A proof of the formula yields a solution (e.g. a program)

Problems:

- Give specifications of computational phenomena (protocols, programming paradigms, program synthesis) in terms of logical formulas.

Research directions

- Computations-as-proofs: synthesis into proof-search.

A specification of a computational problem, esp. synthesis problem (to define a certain function) can be seen as a formula of some logic. A proof of the formula yields a solution (e.g. a program)

Problems:

- Give specifications of computational phenomena (protocols, programming paradigms, program synthesis) in terms of logical formulas.
- Investigate adequate logical formalisms and lambda-calculi and their properties, esp. complexity.

Research directions

- Computations-as-proofs: synthesis into proof-search.

A specification of a computational problem, esp. synthesis problem (to define a certain function) can be seen as a formula of some logic. A proof of the formula yields a solution (e.g. a program)

Problems:

- Give specifications of computational phenomena (protocols, programming paradigms, program synthesis) in terms of logical formulas.
- Investigate adequate logical formalisms and lambda-calculi and their properties, esp. complexity.
- Represent formulas as automata (programs) and conversely. Derive complexity bounds.

Research directions

- Complexity of various logics and their fragments.

Proof construction seen as automaton: a judgment

$$\Gamma \vdash \varphi$$

corresponds to a configuration of an automaton, where:

$$\varphi \quad \approx \quad \text{state}$$

Research directions

- Complexity of various logics and their fragments.

Proof construction seen as automaton: a judgment

$$\Gamma \vdash \varphi$$

corresponds to a configuration of an automaton, where:

$$\begin{aligned} \varphi &\approx \text{state} \\ \Gamma &\approx \text{memory} \end{aligned}$$

Research directions

- Complexity of various logics and their fragments.

Proof construction seen as automaton: a judgment

$$\Gamma \vdash \varphi$$

corresponds to a configuration of an automaton, where:

$$\begin{array}{rcl} \varphi & \approx & \text{state} \\ \Gamma & \approx & \text{memory} \\ \text{proof-search process} & \approx & \text{computation} \end{array}$$

Research directions

- Proof-search games.

The game paradigm (Lorenzen and others):
logic is about *alternating* automata, i.e., games:

- Formula = game.
- Proof = winning strategy of player $\exists$.
- Refutation = winning strategy of player $\forall$.
- Correctness criterion = the existence of strategy.
- Logic = investigating strategies (proof search).

We investigate games for various logics.

Possible projects

Lambda-calculi and such:

- Translations between (typed) lambda-calculus and (typed) Combinatory Logic.
- Computing in the Fujita's polymorphic calculus with $\exists, \wedge$.
- Retractability for simple, polymorphic and intersection types.
- Non-uniform definability in simple types.

Possible projects

Intersection types

- Decidable sub-systems.
- Semantic type-assignment (e.g. Scott).
- Kripke models for intersection logics.

Possible projects

Games:

- Games for classical logic (a big challenge).
- Completeness and normalization via Mezhirov's games.
- Refutation calculi (a big challenge).
- Games for polymorphism.

Possible projects

Complexity of constructive logic.

- Various fragments of first-oder logic.
- And second-order propositional logic.
- Undecidability of linear logic with &, ! and $\multimap$.

Possible projects

Computation as proof: represent various computation-related problems as proof-search (inhabitation) problems

- Classical satisfiability.
- Communication protocols.
- Temporal model-checking.

Possible projects

Predicate logics:

- Fix Tait's donkey-elimination proof.
- Normalization for full second-order propositional logic.

To negotiate a specific project contact me directly at

urzy@mimuw.edu.pl