Categorical Type Theory and its Uses

Valeria de Paiva

School of Computer Science
University of Birmingham, UK

January 2000

Jumping to Conclusions

- Categorical Type Theory: powerful tool, varied applications
  functional programming, authentication, computational linguistics, translation of logical formalisms, etc
- Finding increasingly widespread use
- Requires extensive logical & mathematical background
- But results can be understood independently of CTT.
- Key ideas:
  1. Computation as proof
  2. CTT gives semantics of proofs

Plan of Talk

- Introduction

  The role of formalization in computer science,

  and of categorical type theory in particular

- The basics of categorical type theory:

  combining logic, programming & algebra, to assist in design of type theories

- Applications of CTT:

  — Linear logic & linear type theory

  — Linear functional programming & partial evaluation

  — Linguistics, knowledge representation, authentication

- Conclusions

Introduction: Theory and Computation

- Computation is still not well understood
- Theory links logic, computation & type theory
  - — Programs compute functions
  - — Type theory gives abstract representation of functions
  - — Type theory gives abstract representation of logical proofs
- Two roles of formalization
  - — Verifying correctness / revealing bugs in existing practice
    - e.g. security flaws in Java (Dean et al)
  - — Informing the design of new systems
    - e.g. type checking in Java

The Benefits of Categorical Type Theory (CTT)

- Traditional logic gives semantics of formulas.
- Type theory gives syntax for proofs of formulas.
  - — Why interesting: programs can be viewed as proofs
  - — How is it done: Curry-Howard isomorphism links proofs to types
- CTT gives the semantics of proofs.
  - — Finer grained equivalences between programs
- Gives a powerful tool for understanding and developing e.g.:
  - — programming languages
  - — program generation and optimization
  - — protocols, e.g. authentication protocols
  - — inference preserving translation between logical formalisms
  - — (categorial) grammar formalisms for natural language
  - — novel logics and type theories, e.g. linear logic, linear λ-calculus

The Basics of Categorical Type Theory

Overview:

- Type theory
- Curry-Howard Isomorphism: links type theory to logic & proof
- Category theory and categorical logic
- Categorical type theory

Type Theory

- Programs compute functions

- Abstract representation of functions within the lambda-calculus
  (more generally, within type theory)

- Computation in lambda-calculus is beta-reduction

$$(\lambda x.t)u \;\; \Rightarrow \;\; t[u/x]$$

- All computation in functional languages is beta-reduction
  Most computation in non-functional languages still corresponds to β-reduction
  Type-theoretic extensions to lambda-calculus deal with side effects etc

Curry-Howard Isomorphism: Connecting type theory & proofs

- **Natural deduction proofs** in (constructive, propositional) **logic** can be paired with **terms** in the (simply typed) **lambda-calculus**.

- Pairing formula with λ-term records how formula was proved/compute
  A formula may have different λ-terms if more than one way of proving it.
  Formula gives type of terms/proofs

- Beta-reduction of λ-terms $\equiv$ normalization of ND proofs

- Reduction/normalization collapses computations/proofs which are essentially the same
  Preserves distinctions between significantly different computations/proofs
  (But doesn't always distinguish finely enough; hence CTT)

Illustration: Curry-Howard for Implication

Natural deduction rules for implication (without λ-terms):

$$\frac{A \to B \qquad A}{B} \to_E \qquad\qquad \frac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \to B} \to_I$$

Natural deduction rules for implication (with λ-terms):

$$\frac{M : A \to B \qquad N : A}{M(N) : B} \to_E \qquad\qquad \frac{\begin{array}{c}[x : A] \\ \vdots \\ M : B\end{array}}{\lambda x.M(x) : A \to B} \to_I$$

Function application *λ-abstraction*

Basics of Category Theory

- View objects solely through the relations between them
  Relations (morphisms/maps) between objects matter more than objects

- Leads to generalization of algebra:
  Structures come with structure-preserving maps, e.g.
  sets with *functions*
  groups with *homomorphisms*

- A category $\mathcal{C}$ is a collection of objects $(A, B, \ldots)$ and maps M between them $(f : A \to B, \ldots)$, such that
  (a) there exists an associative composition of maps, $\circ$
  $f : A \to B$, $g : B \to C$ then $g \circ f = h : A \to C$
  (maps are closed under composition: h is in M)
  (b) each object A has an identity map $id_A : A \to A$ such that
  for any $f : A \to B$, $\quad id_A \circ f = f = f \circ id_B$

Categorical Logic/Proof Theory

- Categorical Proof Theory:

  formulas $\Leftrightarrow$ objects in (appropriate) category
  (ND) proofs $\Leftrightarrow$ morphisms in (appropriate) category

  Categorical structure models logical connectives.

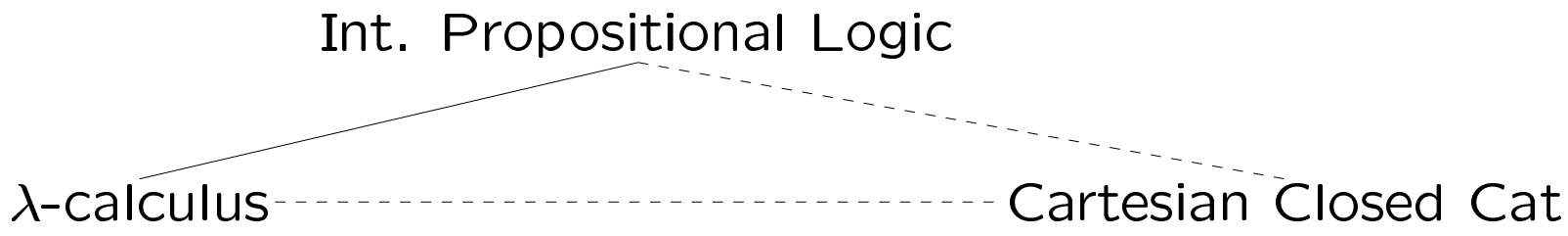
  Hard to find category ensuring isomorphism *Category*$\Leftrightarrow$*Logic*

- Iso *Category*$\Leftrightarrow$*Type* easier to establish (Lambek & Scott 86)

- Use *Category*$\Leftrightarrow$*Type* plus *Type*$\Leftrightarrow$*Logic* (Curry-Howard)
  To induce *Category*$\Leftrightarrow$*Logic*

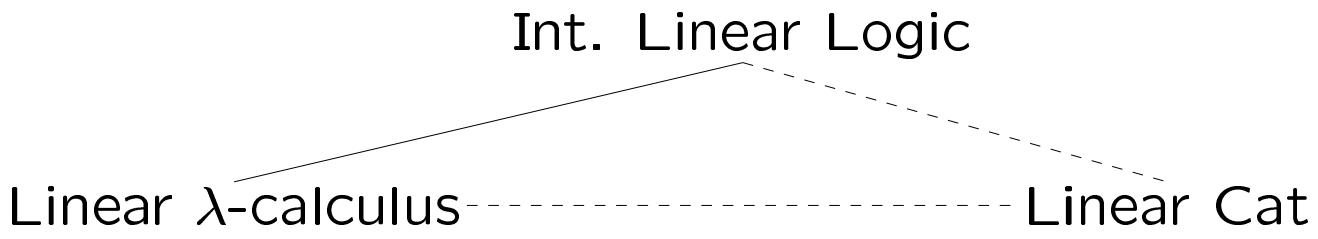
  and hence choose appropriate category

Categorical Type Theory: Extended Curry-Howard Isomorphism



- Proposed by Lawvere, developed by Lambek & Scott 86
- Extensively applied to functional programming
  — better tools for distinguishing proofs
- Extension to Linear Logic & Linear Functional Programming (Paiva et al)

Application of CTT to Linear Logic (Paiva et al)



Three points of the triangle

- Linear Logic
- Linear categories
- Linear type theory

Connections to linear functional programming

Linear Logic

- Recent innovation in logic/computer science (Girard 87)

  Resource sensitive logic

- Unlike traditional logic, formulas are resources that get used up. But the modality ! allows you to relax this, when desired.

Traditional implication:	$A, A \to B \vdash B$	
	$A, A \to B \vdash A \wedge B$	
Linear implication:	$A, A \multimap B \vdash B$	
	$A, A \multimap B \nvdash A \otimes B$	A is used up
With modality:	$!A, A \multimap B \vdash B$	
	$!A, A \multimap B \vdash A \otimes B$	A is duplicated

- Multitude of applications in computer science, linguistics, AI

Linear Categories (Paiva 87,88)

- Work on Gödel's *Dialectica Categories* (Paiva 87)
  Led to first non-algebraic semantics for linear logic that fully preserved distinctions between connectives (Paiva 88).

- Gave additional mathematical foundation to (then) emerging linear logic

- Promoted computational interest in *Chu Spaces* (Barr 79); e.g.
  — concurrency (Brown, Gurr & Paiva 91; Gupta 94),
  — quantum computing (Pratt 97).

- Continuing work on categorical semantics for LL (e.g Paiva 99).

Linear λ-Calculus (Paiva et al 93)

- Failed attempts to develop linear λ-calculus
  — Wadler, Abramsky, Lincoln&Mitchell, Troelstra

- Problem was actually incorrect natural deduction proof rule

Incorrect Rule *leads to* *Invalid Proof*

$$\frac{[!A_1] \ldots [!A_n]}{\begin{array}{c}\vdots \\ B\end{array}}\ !_I \qquad \text{All } A_i \text{ !-ed}$$

$$\frac{C \qquad C \multimap !A_1}{!A_1} \qquad \frac{\ldots \qquad [!A_n]}{\begin{array}{c}\vdots \\ B\end{array}}\ !_I$$

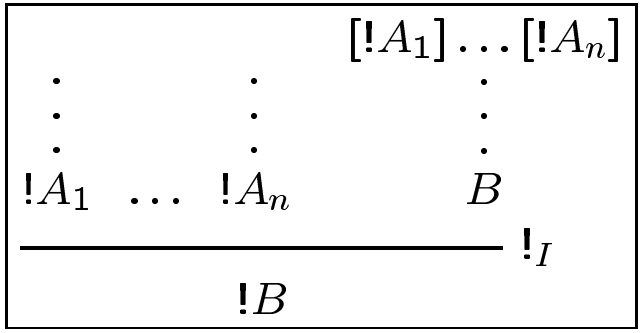
Combines two valid proofs to give invalid proof

- Problem diagnosed & solved via CTT (Paiva et al 93)

Correct Natural Deduction Rule for !

- Prior to solution, repeated attempts and unease amongst researchers about possibility of linear type theory.
  Various ad hoc restrictions proposed.

- Simple correction to proof theory revealed by CTT



- Proof rule reflects finer-grained structure of proofs
  — as motivated by categorical considerations

- *Reviewers comment: "Once you see the solution, you wonder why there was ever a problem"*

- Leads directly to well-behaved linear λ-calculus,
  and opens the way to Linear Functional Programming (LFP)

Categorical Intuitions

- Linear model: a smcc together with a *monoidal* comonad $(!, \epsilon, \delta, m_I, m_{A,B})$ such that every free coalgebra $(!A, \delta)$ has a comonoid structure $(!A, \mathsf{er}, \mathsf{dupl})$. Plus every coalgebra morphism must also be a comonoid morphism.

- Extend traditional assignment of morphims to ND proofs to deal with the ! operator.

- Refined view of proofs (through maps) tells us that

$$\dfrac{\dfrac{!!A}{!!!A \quad !A}}{!!A \quad A}\,!A \qquad\qquad \dfrac{\dfrac{!!!A}{!!A \quad !A}}{!A \quad A}\,!A$$

  correspond to maps $\varepsilon_{!!A}; \delta_{!A}; !(\varepsilon_{!A}); !(\varepsilon_A)$ and $\varepsilon_{!!A}; \varepsilon_{!A}; \delta_A; !(\varepsilon_A)$.

- These are in normal form. Algebra: if maps are equal, must have $\varepsilon_{!!A}; !(\varepsilon_A) = \varepsilon_{!!A}; \varepsilon_{!A}$ hence comonad *idempotent*.

Applications of CTT

- Linear logic and linear type theory
- Linear functional programming and partial evaluation
- Linguistics, knowledge representation, authentication

Linear Functional Programming (LFP)

- Garbage collection, sharing: overheads of functional prog.
  Resource awareness of logic/type theory to reduce overhead
  if term won't be needed again after single use, no need to keep it.

- LFP should provide: storage re-use, no garbage collection for linear types, safe (destructive) in-place update, . . .

- Linear λ-calculus crucial for LFP, and
  Use of explicit substitutions to implement λ-reduction efficiently

- **The xSLAM project** (at Birmingham):
  Design linear λ-calculus with explicit substitutions
  Implement abstract machine for LFP.
  CTT has remained crucial in guiding design decisions

Partial Evaluation of Programs (PE)

- PE: optimize program by compile-time application/specialization to static input.
- Example: parser with inputs from grammar, lexicon, and text
  specialize parser to grammar and lexicon; only text varies at run-time
- Binding-time analysis
  determine parts of the computation evaluable at compile-time & and parts only evaluable at run time.
- (Programmer supplied) type declarations, with type-checking by compiler, to mark partially evaluable computations
- Logical perspective on types (Curry-Howard) largely neglected.

Modal type theory for binding-time analysis

- Two type constructors
  - $\Box A$ code (of type A) has no run-time input
  - A code (of type A) may have run-time input
- (Incorrect) proof rule for $\Box$:

$$\frac{\begin{array}{c}[\Box A_1] \ldots [\Box A_n] \\ \vdots \\ B\end{array}}{\Box B}\ \Box_I$$

— All assumptions A_i must be $\Box$-ed

B constructed purely from code with no run-time input, therefore B has no run-time input, $\Box B$

— *special case when $n = 0$, B has no input of any kind*

- Must prevent invalid glueing of proofs (cf. linear λ-calculus).

Corrected Modal Type Theory

- Problem with glueing proofs / combining code:

$$\dfrac{\dfrac{C \quad C \to \Box A_1}{\Box A_1} \qquad \begin{array}{c}\dots \quad [\Box A_n] \\ \vdots \\ B \end{array}}{\dfrac{}{\Box B}\,\Box_I}$$

  Can construct code without run-time input from code with run-time input!

- Adapt solution from linear λ-calculus:

$$\dfrac{\begin{array}{cccc} & & & [\Box A_1] \dots [\Box A_n] \\ \vdots & & \vdots & \vdots \\ \Box A_1 & \dots & \Box A_n & B \end{array}}{\Box B}\,!_I$$

  Prevents invalid typing judgements

Abstract Machines for Partial Evaluation (ICALP-98)

- Design abstract machine for normalizing type judgments
  separate compile & run-time mixtures into pure run & compile time components, and evaluate them
- Dual Intuitionistic Modal Logic (DIML)
  Reformulation of modal type theory to support this separation, e.g.

$$\frac{\Gamma|\_ \vdash M : A}{\Gamma|\Delta \vdash \Box M : \Box A}\ \Box_I \qquad \frac{\Gamma|\Delta_1 \vdash M : A \quad \Gamma, x : A|\Delta_2 \vdash N : B}{\Gamma|\Delta_1, \Delta_2 \vdash \texttt{Let}\ M\ \texttt{be}\ x\ \texttt{in}\ N : B}\ \Box_E$$

  Contexts divided into $Boxed|Int$ parts

- DIML derived from DILL (Barber&Plotkin)
  contexts divided into !-$ed|Linear$ parts
  developed with extensive use of category theory
- xDIML: add explicit substitutions (cf. xSLAM)
- Abstract machine fully designed
  (Less general) version implemented at Oregon

Further Applications of CTT

- Categorial grammar & computational linguistics

- Translating between logical formalisms

  e.g. for knowledge representation

- Authentication Logics

Categorial Grammar (CG)

- Lambek calculus, underlying categorial grammar, originates from category theory.

- Linear logic perceived as potential theoretical basis of CG. Moortgat 90, Hepple 90: intuitions from LL modalities.

- Paiva 91 proved connection between Lambek calculus and LL via categorical semantics (verifying existing practice)

- Also showed semantically correct way of introducing modalities into CG (guiding further development)

- Connections to LL based glue semantics for NL at PARC
  - Modalities and DILL may be crucial for context-dependency in semantics
  - Provide semantics for packed glue proofs

Inference Preserving Translation

- *Institutions* (Goguen & Burstall 92): categorical method for relating logics using different sets of predicates and constants.
- Kestrel / NASA / SRI use this for program generation
- Applicable to converting between knowledge representation formalisms.
- *Institutions* concerned only with validity of **formulas**
- CTT: extend institutions to deal with validity of **proofs**.
- Potential application to using multiple automated deduction tools (e.g. CAPSL in authentication)

Authentication Logics

- BAN (Burrows, Abadi & Needham) logic seminal in finding bugs in published protocols for key distribution.
  - Axiomatic encoding of intuitions about some existing protocols
  - Ad hoc axiomatic extension to cover new authentication protocols
- Semantics for authentication logics:
  - Independent validation of axioms (soundness, completeness).
  - Plethora of competing Kripke semantics
- Why categorical semantics for authentication logics?
  - Constructive semantics: proving vulnerability also gives example of attack
  - CTT gives correct generalization of alternative Kripke semantics
- CTT for inference-preserving translation
  - Combine authent. logics, model checking & inductive proof techniques:
  - Common Authentication Protocol Specification Language, Millen 97

Conclusions

- Categorical Type Theory: powerful tool, varied applications
  functional programming, authentication, computational linguistics, translation of logical formalisms, etc
- Finding increasingly widespread use
- Requires extensive logical & mathematical background
- But results can be understood independently of CTT.
- Key ideas:
  1. Computation as proof
  2. CTT gives semantics of proofs