ULTRA: Useful Logics, Types, Rewriting, and Automation

Fairouz Kamareddine
Heriot-Watt University

2003-01-31

Overview

Introduction to ULTRA.

Logics, Types, and Rewriting: What and Why.

Specific Research Topics and Future Directions.

A Century of Complexity

	1900	2000
Main way information travels in society:	paper	electric signals, radio
Number of parts in complex machine:	10,000 (locomotive)	1,000,000,000 (CPU)
Worst consequences of single machine failure:	100s die	end of all life?
Likelihood a machine includes a computer:	very low	very high

The Need for Formalism

Because of the increasing interdependency of systems and the faster and more automatic travel of information, failures can have a wide impact. So *correctness* is important.

Modern technological systems are just too complicated for humans to reason about unaided, so *automation* is needed.

Systems have so many possible states that *testing* is often impractical. It seems that *proofs* are needed to cover infinitely many situations.

So *some* kind of formalism is needed to *aid in design* and to *ensure safety*.

What Kind of Formalisms?

reasoning formalism should *at least* be:

Correct: Only correct statements can be "proven".

Adequate: Needed properties in the problem domain can be stated and proved.

Feasible: The resources (money, time) used in stating and proving needed properties must be within practical limits.

What Kind of Formalisms?

suming a minimally acceptable formalism, we would also like it to be:

Efficient: Costs of both the reasoning process *and* the thing being reasoned about should be minimized.

Supportive of reuse: Slight specification changes should not force reproving properties for an entire system. Libraries of pre-proved statements should be well supported.

Elegant: The core of the reasoning formalism should be as simple as possible, to aid in reasoning about the formalism itself.

ULTRA Research Themes

Useful

Logics

Logic is the foundation for rigorous reasoning. There is an ongoing search for better logics and for better methods for verifying the correctness of logics.

Types

Types are a foundation for making logics more flexible without losing correctness and safety. Types are also being used increasingly often for analyzing complex higher-order systems.

Rewriting

Rewriting is using rules of logic, mathematics, or computation in a stepwise manner. Rewriting theory supports reasoning about equivalences between propositions or programs and efficient computation strategies.

and their

Automation

Modern theories of logic, types, and rewriting and the systems to which they are applied have become so complicated that automation is essential.

Applications

Systems of logic, types, and rewriting have applications in the design and implementation of programming languages and theorem provers, in mathematics and in natural language.

Overview

Introduction to ULTRA.

Logics, Types, and Rewriting: What and Why.

Specific Research Topics and Future Directions.

Proofs? Logics? What are they?

A proof is the *guarantee* of some statement provided by a rigorous *explanation* stated using some *logic*.

A logic is a formalism for statements and proofs of statements. A logic usually has *axioms* (statements "for free") and *rules* for combining already proven statements to prove more statements.

For example, this is provable in the logic PROP:

$$A,\ B,\ A \to B \to C \vdash C$$

This is not:

$$A,\ B,\ A \to D \to C \nvdash C$$

Why do we believe the explanation of a proof? Because a proved statement is derived step by step from explicit assumptions using a trusted logic.

Logic is an Area of Active Research

New logics are regularly invented for specialized purposes. Known logics may be *too inflexible* for the task. Or they may be *too flexible*, interfering with automated proof search.

Broken logics are regularly invented. A recent example: The 1988 version of the OCL (Object Constraint Language) sublanguage of UML (Unified Modelling Language) had Russell's paradox of a nearly a century earlier! It is still not known if the revised OCL and/or UML is consistent.

There has been an explosion of new logics in the 20th century. How do we know which ones to trust?

What are Types?

Euclid's *Elements* (circa 325 B.C.) begins with:

1. A *point* is that which has no part;
2. A *line* is breadthless length.

   ⋮

5. A *circle* is a plane figure contained by one line such that all the straight lines falling upon it from one point among those lying within the figure are equal to one another.

Although the above seems to merely *define* points, lines, and circles, more importantly it *distinguishes* between them.

This prevents undesired reasoning, like considering whether two points (instead of two lines) are parallel.

Undesired reasoning? Euclid would have said: *impossible* reasoning. When considering whether objects are parallel, intuition implicitly forced Euclid to think about the *type* of the objects. Because intuition does not support parallel points, Euclid does not even *try* such reasoning.

Why Types are Needed for Logic

Mathematical systems have become less intuitive, for several reasons:

– very complex or abstract
– formal
– Something without intuition is using the system: a computer.

Non-intuitive systems are vulnerable to *paradoxes*. The human brain's built-in type machinery can fail to warn against an impossible situation. Reasoning can proceed obtaining results that may be wrong or paradoxical.

Example: Russell [1902] and Frege [1902] showed that Naive Set Theory had a *paradox*. Let S be "the set of all sets which do not contain themselves". Then, *both* of these are provable:

$$S \in S \qquad\qquad S \notin S$$

Russell [1908] Russell began the use of *types* to solve this problem.

A Quick Introduction to Rewriting

e all know how to do *algebra*:

$$
\begin{array}{rll}
 & (a+b)-a & \text{by rule } x+y=y+x \\
= & (b+a)-a & \text{by rule } x-y=x+(-y) \\
= & (b+a)+(-a) & \text{by rule } (x+y)+z=x+(y+z) \\
= & b+(a+(-a)) & \text{by rule } x+(-x)=0 \\
= & b+0 & \text{by rule } x+0=x \\
= & b &
\end{array}
$$

ewriting is the action of replacing a subexpression which is matched by an instance of one le of a rule by the corresponding instance of the other side of the same rule. If you know gebra, you understand the basics of rewriting.

Important Issues in Rewriting

Orientation: Usually, most rules can only be used from left to right as in $x + 0 \to x$. Forward use of the oriented rules represents progress in computation. Unoriented rules usually do trivial work as in $x + y = y + x$.

Termination: It is desirable to show that rewriting halts, i.e., to avoid infinite sequences of the form $P \to P_1 \to P_2 \to \cdots$.

Confluence: It is desirable that the result of rewriting is independent of the order in the rules are used. For example, $1 + 2 + 3$ should rewrite to 6, no matter how we evaluate it.

Higher-Order Rewriting and Logic

Church's λ-calculus provides *higher-order* rewriting, allowing equations like:

$$f(\underline{(\lambda x.\, x + (1/x))5}) = f(5 + \underline{(1/5)}) = f(\underline{5 + 0.2}) = f(5.2)$$

Church [1940] introduced the simply typed λ-calculus (STLC) and on top of it his Simple Type Theory (CSTT) to provide paradox-free logic. The modern descendant of CSTT is the so-called "higher-order logic" (HOL).

The Convergence of Logics, Types, and Rewriting

Heyting [1934], Kolmogorov [1932], Curry and Feys [1958] (improved by Howard [1980]), and de Bruijn [Nederpelt et al., 1994] all observed the "*propositions as types*" or "*proofs as terms*" (PAT) correspondence.

In PAT, not only is the λ-calculus embedded in the *propositions* as in HOL, but the structure of *proofs* is also given by another level of λ-terms. λ-terms are viewed as proofs of the propositions represented by their types.

Advantages of PAT include:

- better proof manipulation,
- better independent proof checking,
- the extraction of computer programs from proofs, and
- proving the consistency of the logic via the termination of the rewriting system.

Overview

Introduction to ULTRA.

Logics, Types, and Rewriting: What and Why.

Specific Research Topics and Future Directions.

Automated Proving of Termination of Recursive Functions

EPSRC funded. Joint work with Francois Monin at IRISA in France.

Although the termination of recursive functions is an undecidable question, nevertheless theorem provers need to prove the termination of user-supplied recursive functions.

Via the Curry-Howard isomorphism, this has implications for (functional) programming languages (programs are proofs).

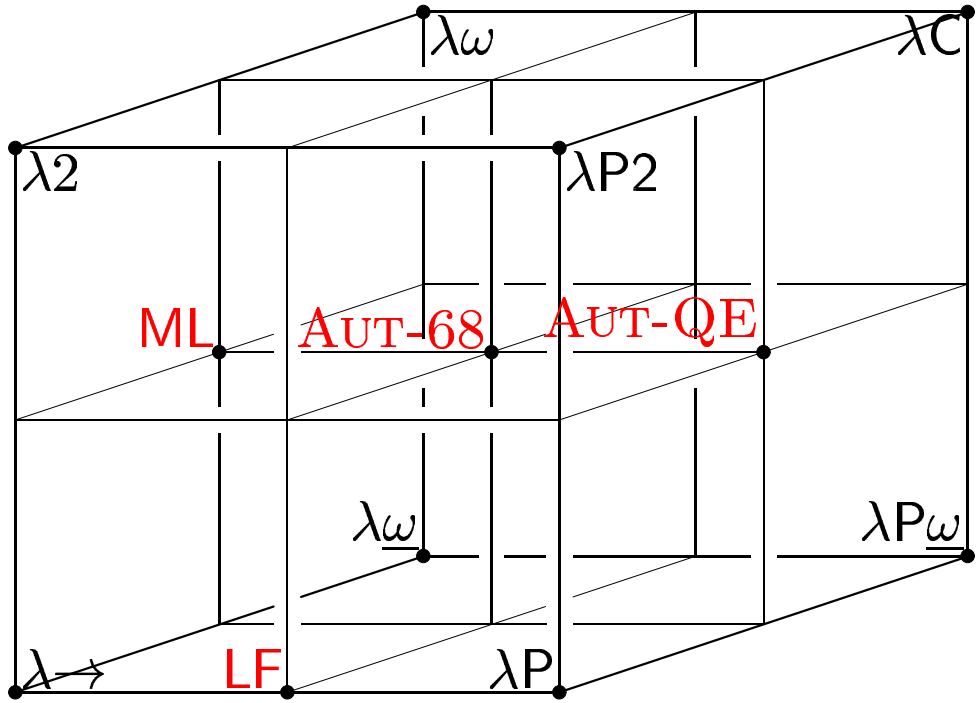
NQTHM requires the user to supply a decreasing measure. Coq establishes the termination fully automatically, but only for a restricted class of functions (structurally inductive).

In Germany and France, proving termination of recursive functions (automatically and non-automatically) is a thriving subject.

New automatable techniques are needed for larger classes of functions.

Precise Systems for Theorem Proving and Prog. Languages

efined the well known Lambda Cube of 8 Pure Type Systems (PTSs) with 19 additional
termediate systems:



nded by the Royal society, the British council and the Dutch research council (NWO).
int work with Rob Nederpelt and Twan Laan in the Netherlands.

This provides logics with *exactly* the needed reasoning power for the task at hand. This can help both with automation as well as with proving a logic to be correct.

Some previous systems correspond to some of the new systems, giving better classifications and improving our understanding.

Termination of Entire Systems

Joint work with: Alejandro Ríos in Argentina and Roel Bloo in the Netherlands.

Funded by EPSRC and the Dutch research councils (NWO and SION).

Although termination is always useful and is essential for proving consistency of PAT-based logics, in general it is undecidable.

Weak normalization involves showing termination for a specific strategy of choosing rewrite steps. The hard part is finding the strategy.

New methods are needed for proving termination and finding terminating strategies.

The termination of subsystems of λs_e remains open despite many efforts in NL, France and UK.

Automation of these methods is also a hot topic and libraries of automated proofs are being developed in the USA, France and the UK.

With Qiao Hayan of Gothenberg, Sweden, we have automated the proofs of termination of two entire calculi:
the French σ-calculus and my own s-calculus in the Swedish theorem prover ALF.

(Higher-Order) Unification via Explicit Substitution

Funded by EPSRC and the Brasilian funding body CNPq.

Joint work with Mauricio Ayala-Rincon and Flavio de Moura at University of Brasilia in Brasil. Based on work:

– In France: the Coq and rewriting teams in INRIA, universities of Paris and Nancy, France.

Robinson's resolution principle was extended to other settings such as the typed λ-calculus (Huet). This is essential for automated deduction in higher order logics. Several Higher Order Unification (HOU) approaches have been developed and used in PLs and TPs (e.g., λ-prolog and Isabelle).

HOU is undecidable and involves substitution (which is tricky in the λ-calculus).

We need HOU methods that are useful for deduction in typed λ-calculus and other Higher-Order systems.

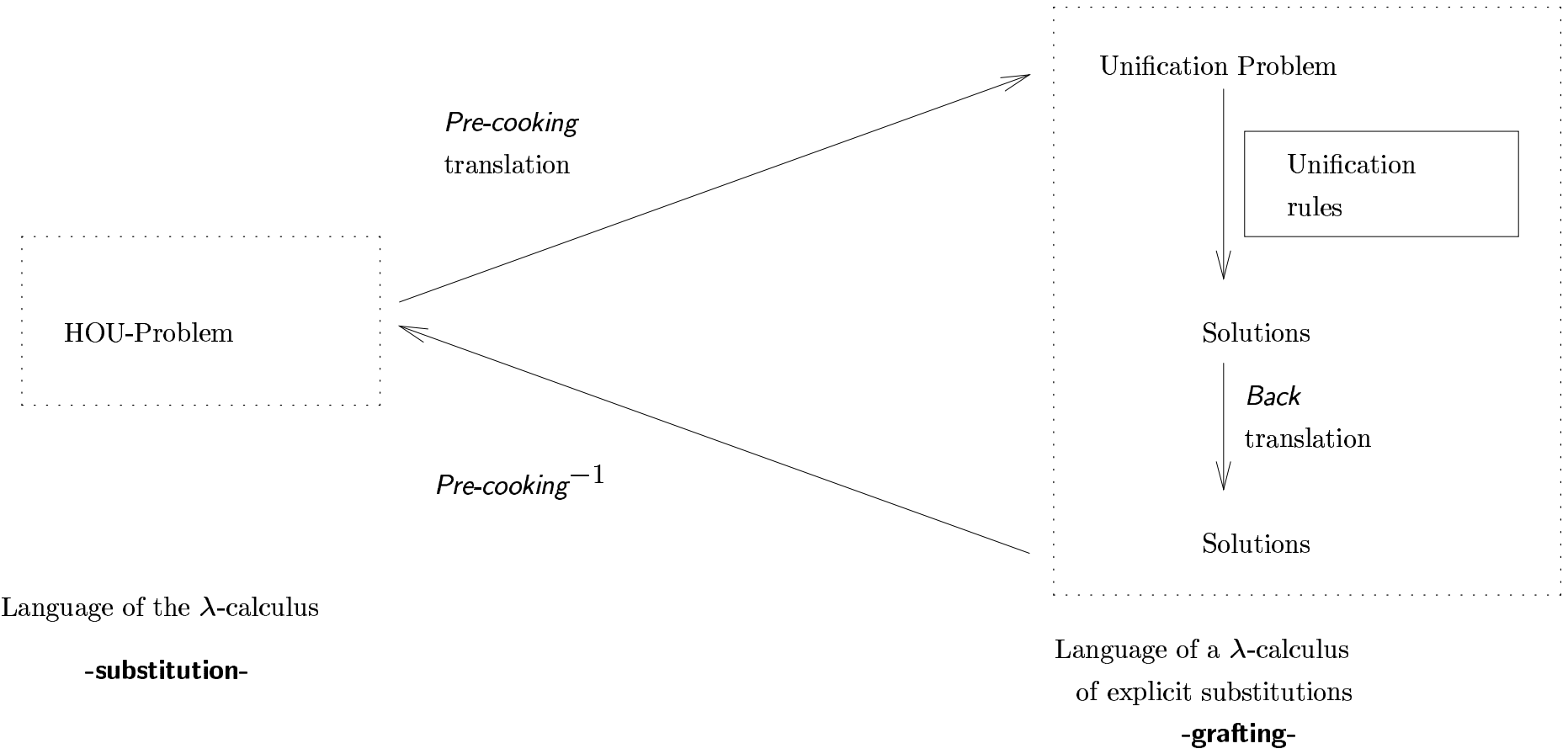


Figure 1: HOU method via calculi of explicit substitutions

Formalising and automating the natural language of mathematics

Joint work with Rob Nederpelt in Eindhoven, the Netherlands.

Funded by EU through a network of excellence involving Italy, France, UK, NL, Austria and Germany, with collaborations from USA and Canada.

Mathematics is written *in natural* language: *ambiguous*, *incomplete*, *poorly organised*.

Mathematicians have rarely became interested in the huge amount of work of *automating* mathematics (e.g., Automath) and of *formalising* it (e.g., Russell, Whitehead, Frege).

Existing automated systems are *complex*. Mathematicians *do not* see their advantages.

The challenge is to write mathematics in a formal automatable way which engages mathematicians and promotes collaboration with computer scientists.

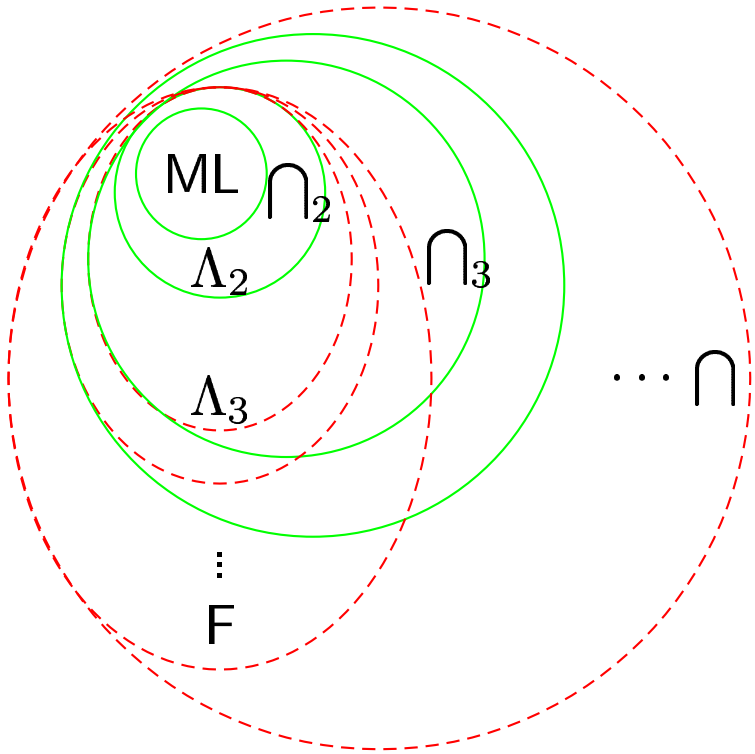
This enables:

I. *Computer assistance* in the *development* of mathematics.

II. *Computerized verification* of mathematical theories.

More Powerful Type Analysis for Programming Languages

F: System F.
Λ_k: rank-k System F.
$\bigcap$: intersection types.
$\bigcap_k$: rank-k of $\bigcap$.
Decidable.
Undecidable.



. Wells has:

Established the undecidability of System F and its finite-rank restrictions.

Proved decidability of finite-rank restrictions of intersection types at rank 3 and above.

Developed the first understandable analysis algorithms for the systems of intersection types at rank 3 and above.

Supported by EPSRC, NSF and EU funding to develop these ideas for practical use.

onzo Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

B. Curry and R. Feys. *Combinatory Logic I*. Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1958.

Frege. Letter to Russell. English translation in [Heijenoort, 1967], pages 127–128, 1902.

van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, Cambridge, Massachusetts, 1967.

Heyting. *Mathematische Grundlagenforschung. Intuitionismus. Beweistheorie.* Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer-Verlag, Berlin, 1934.

. A. Howard. The formulaes-as-types notion of construction. In J. R[oger] Hindley and J[onathan] P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980. ISBN 0-12-349050-2. An earlier version was privately circulated in 1969.

N. Kolmogorov. Zur Deutung der Intuitionistischen Logik. *Mathematisches Zeitschrift*, 35:58–65, 1932.

ob Nederpelt, J. H. Geuvers, and Roel C. de Vrijer. *Selected Papers on Automath*. North-Holland, Amsterdam, 1994.

Russell. Letter to Frege. English translation in [Heijenoort, 1967], pages 124–125, 1902.

Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908. Also in [Heijenoort, 1967], pages 150–182.