ULTRA: Useful Logics, Types, Rewriting, and Applications

Prof. Fairouz Kamareddine
Dept. of Computing & Electrical Engineering
Heriot-Watt University

2 February 2000

Heriot-Watt University: Inaugural Lectures 2000

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 Some Kind of 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?

A 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?

Assuming 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.

Logics, Types, and Rewriting

Logics, types, and rewriting are

- elegant, as we can formulate and (automate) clear rules of how they work (e.g., from A and $A \to B$ we can deduce B),
- adequate (we can express a lot in these tiny formalisms), and
- able to be shown correct.

Logics, types, and rewriting have existed in various since from the times of the ancient Babylonians and Greeks (e.g., Euclid, Aristotle, etc.).

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.
- 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.
- There has been an explosion of new logics in the 20th century. How do we know which ones to trust? Fund us and we will tell you ...

A Brief History of Logic (Aristotle)

- Aristotle (384–322 B.C.) wanted a set of rules that would be powerful enough for most intuitively valid proofs.

- Aristotle correctly stated that *proof search* is harder than *proof checking*:

  Given a proof of a statement, one can check that it is a correct proof.
  Given a statement, one may not be able to find the proof.

  Aristotle's intuitions on this have been confirmed by Gödel, Turing, and others.

A Brief History of Logic (Leibniz)

- Leibniz (1646–1717) conceived of *automated deduction*, i.e., to find
  - a language L in which arbitrary concepts could be formulated, and
  - a machine to determine the correctness of statements in L.
- Such a machine can not work for every statement according to Aristotle and (later results by) Gödel and Turing.

A Brief History of Logic (Cantor, Peano, Frege)

The late 1800s saw the beginnings of serious formalization:

- Cantor began formalizing set theory [1, 2] and made contributions to number theory.
- Peano formalized arithmetic [14], but did not treat logic or quantification.
- Frege's *Begriffsschrift* [5] (1879) was the first formalisation of logic which presented logical concepts via symbols rather than natural language. Frege's *Grundgesetze der Arithmetik* [6, 8], called later by others Naive Set Theory (NST), could handle elementary arithmetic, set theory, logic, and quantification.

A Brief History of Logic (Frege's Set Theory)

- Frege's NST allowed a precise definition of the vital concept of the *function*. As a result, NST could include not only functions that take numbers as arguments and return numbers as results, but also functions that can take and return other sorts of arguments, *including functions*. These powerful functions were the key to the formalization of logic in NST.

- Frege was cautious: ordinary functions could only take "objects" as arguments, not other functions. However, to gain important expressive power, he allowed a way to turn a function into an object representing its graph.

- Unfortunately, this led to a *paradox*, due to the implicit possibility of *self-application* of functions.

A Brief History of Logic (Russell's Paradox)

- In 1902, Russell suggested [15] and Frege completed the argument [7] that a *paradox* could occur in NST. First, one can define S to be "the set of all sets which do not contain themselves". Then, one can prove *both* of these statements in NST:

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

- In fact, the same paradox could be encoded in the systems of Cantor and Peano. As a result, all three systems were *inconsistent* — not only could every true statement be proved but also every false one! (Three-valued logic can solve this, but is unsatisfactory for other reasons.) Logic was in a *crisis*.

- In 1908, Russell suggested the use of *types* to solve the problem [16].

A Brief History of Types (Euclid)

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

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

     ⋮

  15. 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, it shows more importantly that Euclid *distinguished* between them. Euclid always mentioned to which *class* (points, lines, etc.) an object belonged.

A Brief History of Types (Euclid)

- By distinguishing classes of objects, Euclid prevented undesired situations, like considering whether two points (instead of two lines) are parallel.

- *Undesired* results? Euclid himself would probably have said: *impossible* results. When considering whether two objects were parallel, intuition implicitly forced him to think about the *type* of the objects. As intuition does not support the notion of parallel points, he did not even *try* to undertake such a construction.

- In this manner, types have always been present in mathematics, although they were not noticed explicitly until the late 1800s. If you have studied geometry, then you have some (implicit) understanding of types.

A Brief History of Types (Paradox Threats)

- Starting in the 1800s, mathematical systems became less intuitive, for several reasons:
  - Very complex or abstract systems.
  - Formal systems.
  - Something with less intuition than a human using the systems: a computer.

- These situations are *paradox threats*. An example is Frege's NST. In such cases, there is not enough intuition to activate the (implicit) type theory to warn against an impossible situation. Reasoning proceeds within the impossible situation and then obtains a result that may be wrong or paradoxical.

Example Failures due to Type Errors

An untyped computer program may receive instructions from a first-year student to add the number 3 to the word "four" (instead of the number 4). The computer is unaware that "four" is not a number and the result of $3 +$ "four" is unpredictable. The computer may

- give an answer that is clearly wrong (for example, true),
- give no answer at all, or
- give an answer that is not so clearly wrong (for example, 6).

Especially the last situation is highly undesirable.

A Brief History of Types (Russell)

- To avoid the paradoxes of the systems of Cantor, Peano, and Frege, Russell prescribed avoiding self-reference and self-application in his "vicious circle principle":

  Whatever involves all *of a collection must not be one of the collection.*

- Russell implemented this in his Ramified Theory of Types (RTT) [16] which used *types* and *orders*. Self-application was prevented by forcing functions of order k to be applied only to arguments of order less than k.

- This was carried out further by Russell and Whitehead in the famous *Principia Mathematica* [17] (1910-1912), which founded mathematics on logic, as far as possible, avoiding paradoxes.

A Brief History of Types (Russell)

- For example, in RTT, one can define a function "+" which is restricted to be applied only to integers.

- Although RTT was correct, unlike NST, the types of RTT have turned out instead to be *too restrictive* for mathematics and computer science where fixed points (to mention one example) play an important role. RTT also forces duplication of the definitions of the number system, the boolean algebra, etc., at *every* level.

- The exploration of the middle ground between these two extremes has led to many systems, most of them in the context of the λ-calculus, the first higher-order *rewriting* system.

A Quick Introduction to Rewriting

We all know how to do *algebra*:

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

Rewriting is the action of replacing a subexpression which is matched by an instance of one side of a rule by the corresponding instance of the other side of the same rule. If you have studied algebra, then you are skilled at rewriting.

Important Issues in Rewriting

- *Orientation*: Usually, most rules can only be used from left to right as in $x + 0 \rightarrow x$. 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 $P \rightarrow P_1 \rightarrow P_2 \rightarrow \cdots$.

- *Confluence*: 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.

A Brief History of Rewriting (Ancients)

- When the Greeks introduced logic they did not have modern-style rewriting.
- The Babylonians on the other hand, developed techniques for symbolic computations through their work on algebra. This can be viewed as rewriting.
- The Arabs of course first introduced algebra in close to its modern form.

A Brief History of Rewriting (λ-Calculus)

- In the late 1800s, Frege identified the *abstraction principle*: Any expression mentioning some symbol in zero or more places can be turned into a function by abstracting over that symbol.

- Introduced in the 1930s, Church's λ-calculus made function abstraction an *operator*. For example, $(\lambda x.\ x + 5)$ represents the (unnamed) mathematical function which takes as input any number and returns as output the result of adding 5 to that number.

- The λ-calculus provides *higher-order* rewriting, allowing equations like:

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

A Brief History of Rewriting (λ-Calculus)

- The type-free λ-calculus, which can be seen as a small computer programming language, is an excellent theory of functions — it can represent all computable functions.

- Church intended the *type-free* λ-calculus with logical operators to provide a foundation for mathematics. Unfortunately, Russell's paradox could also be encoded in the type-free λ-calculus, rendering its use for logic incorrect.

- Church introduced the simply typed λ-calculus (STLC) [3] to provide logic while avoiding Russell's paradox in a manner similar to RTT. Unfortunately, like RTT, the STLC is too restrictive. A modern, slightly less restrictive descendant of this approach is the so-called "higher-order logic" (HOL).

The Convergence of Logics, Types, and Rewriting

- Heyting [10], Kolmogorov [12], Curry and Feys [4] (improved by Howard [11]), and de Bruijn [13] all observed the "*propositions as types*" or "*proofs as terms*" (PAT) correspondence.

- In PAT, logical operators are embedded in the types of λ-terms rather than in the terms and λ-terms are viewed as proofs of the propositions represented by their types.

- Advantages of PAT include the ability to manipulate proofs, easier support for independent proof checking, the possibility of the extraction of computer programs from proofs, and the ability to prove the consistency of the logic via the termination of the rewriting system.

My Work: Item Notation

- *Item notation* (similar to the notation of de Bruijn's AUTOMATH) writes function abstraction as $(\lambda_x)M$ instead of $(\lambda x.\, M)$ and function application as $(N\delta)M$ instead of $M\,N$.

- Some of my work has explored some of the huge number of technical advantages of item notation over Church's notation, too many to list here. If you are using Church's notation, then you should *immediately* switch to item notation.

- In addition, I have obtained a variety of results using item notation which would have been much more difficult to find otherwise, e.g., various results with explicit substitution, also an extension of $\rightarrow_\beta$ which is confluent and conserves strong normalization.

My Work: Implicit Redexes

- In the λ-term $(n\delta)(+\delta)(\lambda_f)(m\delta)(\lambda_x)(\lambda_y)(y\delta)(x\delta)f$, the pairs $(+\delta)(\lambda_f)$ and $(m\delta)(\lambda_x)$ are β-redexes. The pair $(n\delta)(\lambda_y)$ is an *implicit β-redex*, which would be revealed by contracting the two explicit redexes.

- It is quite desirable to have the option of contracting implicit redexes directly (generalized β-reduction). Also, there are simple transformations which expose implicit β-redexes without contracting any β-redexes, corresponding to permutative conversions of logic and having connections with the CPS transformation and lazy evaluation.

- My work has proven a number of useful properties of generalized β-reduction and transformations which expose implicit β-redexes.

My Work: Definitions

- Many type theories allow certain steps if two types or terms are convertible. It is essential for practical use of these systems (e.g., in theorem prover implementations such as Nuprl, Coq, Lego, etc.) to be able to use definitions in the context in deciding these questions of convertibility.

- For example, access to definitions is needed to be able to show that $(a\delta)(\lambda_{x:*})(\lambda_{y:x})(\lambda_{f:a\to a})(y\delta)f$ is typable, because the knowledge that x is an abbreviation for a is not usable in typing the subexpression $(y\delta)f$.

- My work has proven many important properties of type systems with definitions.

My Work: Explicit Substitutions

- Systems of *explicit substitution* bridge the gap between the meta-theory of substitution and binding and the steps needed to implement these concepts.

- My work has provided explicit substitution calculi having many desirable properties. They (1) simulate one step β-reduction, (2) are confluent (on closed terms), (3) preserve strong normalisation (have PSN), (4) have associated calculi of substitutions that are SN, (5) have simply typed versions that are SN, (6) possess confluent extensions on open terms which have WN.

References

[1] G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (Erster Artikel). *Mathematische Annalen*, 46:481–512, 1895.

[2] G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (Zweiter Artikel). *Mathematische Annalen*, 49:207–246, 1897.

[3] A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

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

[5] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. Also in [9], pages 1–82.

[6] G. Frege. *Grundgesetze der Arithmetik, begriffschriftlich abgeleitet*, volume I. Pohle, Jena, 1892. Reprinted 1962 (Olms, Hildesheim).

[7] G. Frege. Letter to Russell. English translation in [9], pages 127–128, 1902.

[8] G. Frege. *Grundgesetze der Arithmetik, begriffschriftlich abgeleitet*, volume II. Pohle, Jena, 1903. Reprinted 1962 (Olms, Hildesheim).

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

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

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

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

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

[14] G. Peano. *Arithmetices principia, nova methodo exposita*. Bocca, Turin, 1889. English translation in [9], pages 83–97.

[15] B. Russell. Letter to Frege. English translation in [9], pages 124–125, 1902.

[16] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908. Also in [9], pages 150–182.

[17] A.N. Whitehead and B. Russell. *Principia Mathematica*, volume I, II, III. Cambridge University Press, 1910^1, 1927^2. All references are to the first volume, unless otherwise stated.