Proof-Carrying Code:

From 19th Century Logic
To 21st Century Computing

Nadeem Abdul Hamid

Yale University

February 17, 2004

A Bit of Trivia

A Bit of Trivia

triv·i·um *n. pl.* **triv·i·a** [tri-,*three*- + via,*road,way*] The lower division of the seven liberal arts in medieval schools, consisting of grammar, rhetoric, and ***logic*** – being a triple way, as it were, to eloquence.

A Bit of Trivia

triv·i·um *n. pl.* **triv·i·a** [tri-,*three*- + via,*road,way*] The lower division of the seven liberal arts in medieval schools, consisting of grammar, rhetoric, and ***logic*** – being a triple way, as it were, to eloquence.

- What is LOGIC?

A Bit of Trivia

triv·i·um *n. pl.* **triv·i·a** [tri-,*three*- + via,*road,way*] The lower division of the seven liberal arts in medieval schools, consisting of grammar, rhetoric, and ***logic*** – being a triple way, as it were, to eloquence.

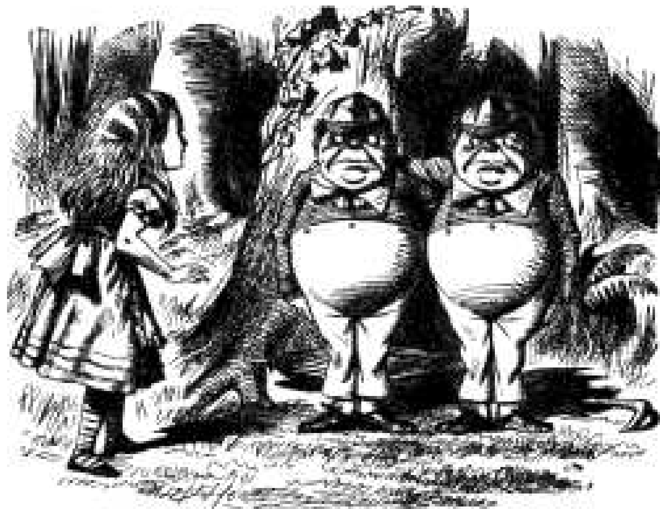
- What is LOGIC?

Charles Lutwidge Dodgson (1832-1898) . . .

Logic!

“Contrariwise,” continued Tweedledee, “if it was so, it might be; and if it were so, it would be; but as it isn’t, it ain’t. That’s logic.”

[Lewis Carroll, *Through the Looking Glass*]



The First Age of Logic

- Symbolic logic (500BC - 19th Century)

– All Greeks are men.

– All men are mortal.

$\rightarrow$ All Greeks are mortal.

The First Age of Logic

- Symbolic logic (500BC - 19th Century)

– All *toves* are *bojums*.

– All *bojums* are *slithy*.

$\rightarrow$ All *toves* are *slithy*.

The First Age of Logic

- Symbolic logic (500BC - 19th Century)

– All *toves* are *bojums*.

– All *bojums* are *slithy*.

$\rightarrow$ All *toves* are *slithy*.

- Paradoxes: “This sentence is false.”

The Age of Mathematical Logic

- Mid–19th to mid–20th century
- Logic as a (*the*) language for mathematics
- Boole, Frege, Cantor, Hilbert, Russell, Gödel, Church, Turing

The Age of Mathematical Logic

- Mid–19th to mid–20th century
- Logic as a (*the*) language for mathematics
- Boole, Frege, Cantor, Hilbert, Russell, Gödel, Church, Turing
- Paradoxes again (naïve set theory: $T = \{S | S \notin S\}$ – is T a member of itself?)
- Incompleteness . . .

The New Age of Logic

Logic in Computer Science

- Boolean circuits
- Combinatorial analysis (NP-completeness)
- Databases: SQL (standard first-order logic)
- Formal semantics (programming languages)
- Design validation and verification (hardware)
- AI: mechanized reasoning and expert systems
- Network security: proof-carrying code

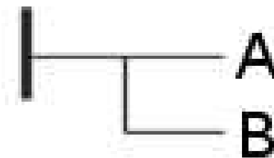
Back to Frege

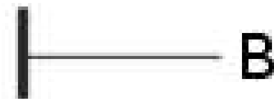
- Gottlob Frege (1848-1925)
- *Begriffschrift* ("Conceptual Notation"), 1879

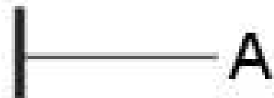
If *today is Tuesday* then *we are in Georgia*.

Today is Tuesday.

We are in Georgia.







Begriffschrift, 1879

New Notation

$$B \to A$$

"B implies A"

$$\frac{\vdash B \to A \qquad \vdash B}{\vdash A}$$

(*modus ponens*)

$$\vdash A \to A \tag{1}$$

$$\vdash A \to (B \to A) \tag{2}$$

$$\vdash (A \to (B \to C)) \to ((A \to B) \to (A \to C)) \tag{3}$$

More Logical Connectives, Quantifiers

A implies B	$A \to B$
A and B	$A \wedge B$
A or B	$A \vee B$
not A	$\neg A$
for all ...	$\forall \ldots$
$\vdots$	

Natural Deduction

- Gerhard Gentzen, 1934
- Use of assumptions: $B_1, ...B_n \vdash A$
  $(\Delta, \Gamma = B_1, ...B_n)$

Natural Deduction

- Gerhard Gentzen, 1934
- Use of assumptions: $B_1, ...B_n \vdash A$
  $(\Delta, \Gamma = B_1, ...B_n)$

$$\frac{}{\Delta, A \vdash A}\ \text{ID}$$

$$\frac{\Delta, B \vdash A}{\Delta \vdash B \rightarrow A}\ \rightarrow\text{-I} \qquad \frac{\Delta \vdash B \rightarrow A \qquad \Gamma \vdash B}{\Delta, \Gamma \vdash A}\ \rightarrow\text{-E}$$

$$\frac{\Delta \vdash A \qquad \Gamma \vdash B}{\Delta, \Gamma \vdash A \wedge B}\ \wedge\text{-I} \qquad \frac{\Delta \vdash A \wedge B}{\Delta \vdash A}\ \wedge\text{-E1} \qquad \frac{\Delta \vdash A \wedge B}{\Delta \vdash B}\ \wedge\text{-E2}$$

A Roundabout Proof of $A \wedge B$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{\dfrac{}{(B \wedge A) \vdash (B \wedge A)}\;\text{ID}}{(B \wedge A) \vdash A}\;\wedge\text{-E2}
\qquad
\dfrac{\dfrac{}{(B \wedge A) \vdash (B \wedge A)}\;\text{ID}}{(B \wedge A) \vdash B}\;\wedge\text{-E1}
}{(B \wedge A) \vdash A \wedge B}\;\wedge\text{-I}
}{\vdash (B \wedge A) \rightarrow (A \wedge B)}\;\rightarrow\text{-I}
\qquad
\dfrac{\dfrac{}{B \vdash B}\;\text{ID} \qquad \dfrac{}{A \vdash A}\;\text{ID}}{A, B \vdash B \wedge A}\;\wedge\text{-}
}{A, B \vdash A \wedge B}\;\rightarrow\text{-E}
$$

Alonzo Church, 1903-1995

- Lambda calculus: 1932
- New formulation of logic
- Church-Turing thesis: problems are either solvable or unsolvable by mechanical methods of computation

The Lambda (λ) Calculus

- All computation reduced to notion of substitution
- Compact notation for writing functions

The function f *where* $f(x) = x \times x$.

$$\Rightarrow$$

$$\lambda x.x \times x$$

$$f(3) = 3 \times 3 = 9$$

$$\Rightarrow$$

$$(\lambda x.x \times x)(3) \Rightarrow (x \times x)[3/x] \Rightarrow 3 \times 3 \Rightarrow 9$$

Church-Rosser Theorem

- Reduction order does not matter

$$(\lambda x.x \times x)((\lambda y.y + 1)(2))$$

Church-Rosser Theorem

- Reduction order does not matter

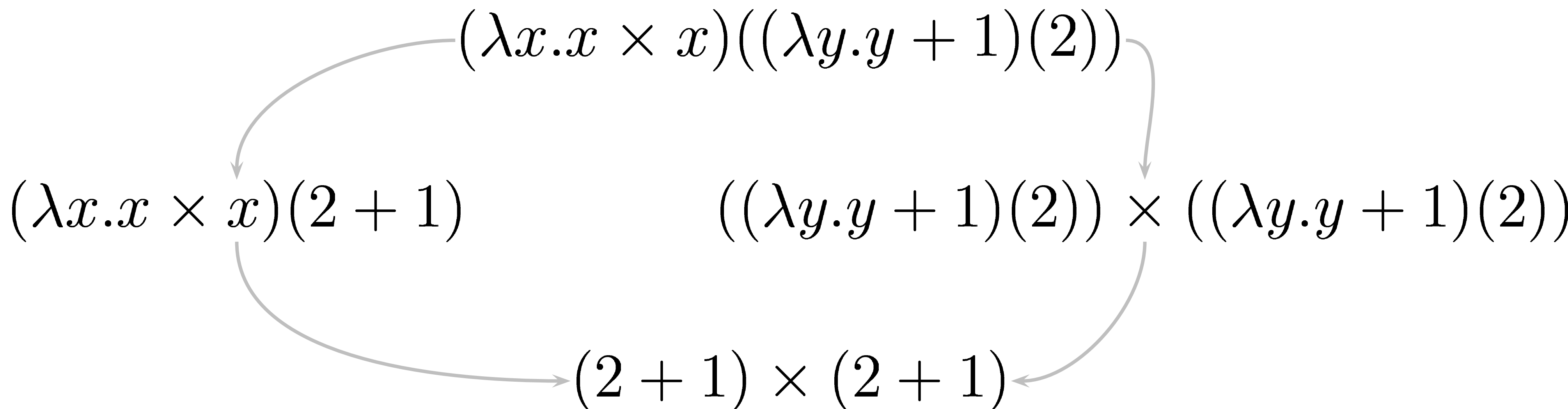
$$(\lambda x.x \times x)((\lambda y.y + 1)(2))$$

$$(\lambda x.x \times x)(2 + 1)$$

$$((\lambda y.y + 1)(2)) \times ((\lambda y.y + 1)(2))$$

Church-Rosser Theorem

- Reduction order does not matter



Curying Multiple Arguments

- Functions return functions

$$g(x,y) = x \times x + y \times y$$
$$g(3,4) = 3 \times 3 + 4 \times 4 = 25$$

$$\begin{aligned} & ((\lambda x.\lambda y.x \times x + y \times y)(3))(4) \\ \Rightarrow\; & (\lambda y.3 \times 3 + y \times y)(4) \\ \Rightarrow\; & 3 \times 3 + 4 \times 4 \\ \Rightarrow\; & 25 \end{aligned}$$

Data Structures: Pairs

$\langle t, u \rangle$ *build a pair*

$\langle t, u \rangle.\textit{fst} \Rightarrow t$ *(reduction*

$\langle t, u \rangle.\textit{snd} \Rightarrow u$ *rules)*

Data Structures: Pairs

$\langle t, u \rangle$ *build a pair*

$\langle t, u \rangle.\mathit{fst} \Rightarrow t$ *(reduction*

$\langle t, u \rangle.\mathit{snd} \Rightarrow u$ *rules)*

$\lambda z.\langle z.\mathit{snd}, z.\mathit{fst} \rangle$ *swap elements*

Data Structures: Pairs

$\langle t, u \rangle$ *build a pair*

$\langle t, u \rangle.\textit{fst} \Rightarrow t$ *(reduction*

$\langle t, u \rangle.\textit{snd} \Rightarrow u$ *rules)*

$\lambda z.\langle z.\textit{snd}, z.\textit{fst} \rangle$ *swap elements*

$$
\begin{aligned}
& (\lambda z.\langle z.\textit{snd}, z.\textit{fst} \rangle)(\langle y, x \rangle) \\
\Rightarrow \; & \langle \langle y, x \rangle.\textit{snd}, \langle y, x \rangle.\textit{fst} \rangle \\
\Rightarrow \; & \langle x, y \rangle
\end{aligned}
$$

No Extensions Needed!

- Numbers: $n = \lambda f.\lambda x.f(\cdots f(x))$
- Addition, multiplication, subtraction, ...
- Booleans, pairs, other data structures, ...
- Recursive functions

- *Any function on numbers computable by a machine can be represented by a lambda term, built from (1) $\lambda x.t$, (2) $t(u)$, and (3) x!*

Typed Lambda Calculus

- Introduced 1940, to avoid paradoxes

- Function types: $A \to B$
- Type of a pair: $A \wedge B$
- Typing judgment: $x_1 : B_1, \ldots x_n : B_n \vdash t : A$

- Function application rule:

$$\frac{\Delta \vdash t : B \to A \qquad \Delta \vdash u : B}{\Delta \vdash t(u) : A}$$

λ Typing Rules

$$\frac{}{x : A \vdash x : A}\ \text{ID}$$

$$\frac{\Delta, x : B \vdash t : A}{\Delta \vdash \lambda x.t : B \to A}\ \to\!-\text{I} \qquad \frac{\Delta \vdash t : B \to A \qquad \Delta \vdash u : B}{\Delta \vdash t(u) : A}\ \to\!-\text{E}$$

$$\frac{\Delta \vdash t : A \qquad \Delta \vdash u : B}{\Delta \vdash \langle t, u \rangle : A \wedge B}\ \wedge\!-\text{I} \qquad \frac{\Delta \vdash t : A \wedge B}{\Delta \vdash t.\mathsf{fst} : A}\ \wedge\!-\text{E1} \qquad \frac{\Delta \vdash t : A \wedge B}{\Delta \vdash t.\mathsf{snd} : B}\ \wedge\!-\text{E2}$$

Type of "pair-swap"

$$\dfrac{\dfrac{\dfrac{}{z : (B \wedge A) \vdash z : (B \wedge A)}\ \text{ID}}{z : (B \wedge A) \vdash z.\mathsf{snd} : A}\ \wedge\text{–E2} \qquad \dfrac{\dfrac{}{z : (B \wedge A) \vdash z : (B \wedge A)}\ \text{ID}}{z : (B \wedge A) \vdash z.\mathsf{fst} : B}\ \wedge\text{–E1}}{\dfrac{z : (B \wedge A) \vdash \langle z.\mathsf{snd}, z.\mathsf{fst} \rangle : A \wedge B}{\vdash \lambda z.\langle z.\mathsf{snd}, z.\mathsf{fst} \rangle : (B \wedge A) \to (A \wedge B)}\ \to\text{–I}}\ \wedge\text{–I}$$

$$\dfrac{\lambda z.\langle z.\mathsf{snd}, z.\mathsf{fst} \rangle : (B \wedge A) \to (A \wedge B) \qquad x : A, y : B \vdash \langle y, x \rangle : B \wedge A}{x : A, y : B \vdash (\lambda z.\langle z.\mathsf{snd}, z.\mathsf{fst} \rangle)(\langle y, x \rangle) : A \wedge B}\ \to\text{–E}$$

The Curry-Howard Isomorphism

Correspondence between natural deduction and the lambda calculus

- First noted by Haskell Curry (1900-1982) in 1960
- Written down by William Howard in 1969
- Finally published in 1980

Proofs are Programs!

$\lambda \rightarrow$	N.D.
variable	assumption
type constructor	connective
type	proposition
term (program)	proof
inhabitation	provability
reducable expression	redundant proof tree
reduction	normalization

Ideas and observations about logic are ideas and observations about programming languages

Proof-Carrying Code

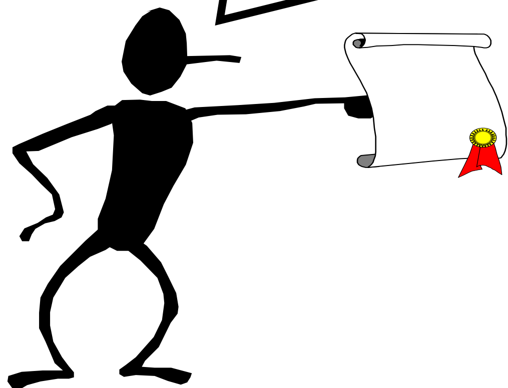
Proofs, Types, and Safe Mobile Code (2004)

- PCC: Necula and Lee, 1996–98
- Foundational PCC: Appel and Felty, 2000–02
- Syntactic FPCC: Hamid, Shao, *et al.*, 2002
- Ongoing work at Berkeley, CMU, Princeton, Yale, . . .

A Logical Approach

Please install and execute this program.

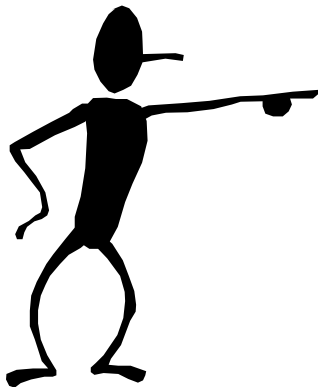
OK, but let me quickly look over the instructions first.

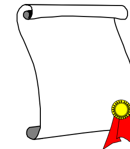


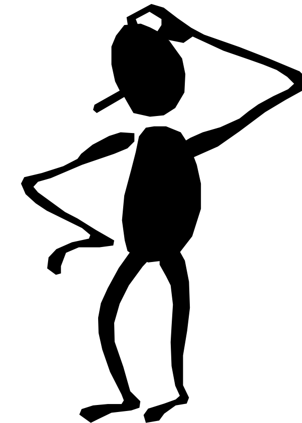
Code producer

Host

A Logical Approach







Code producer

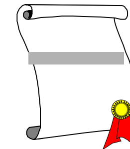
Host

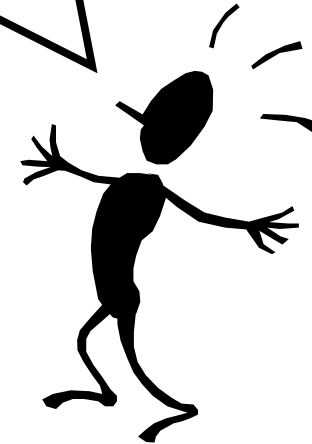
A Logical Approach

Code producer

Host

A Logical Approach

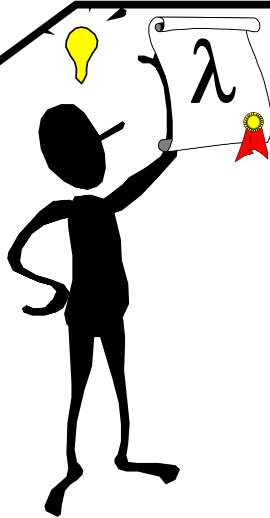


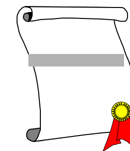


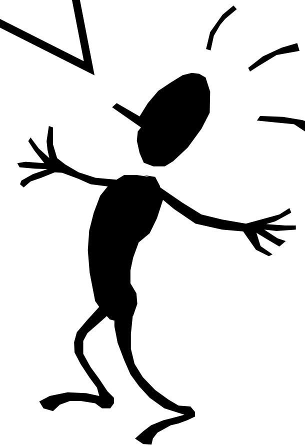
Code producer

Host

A Logical Approach



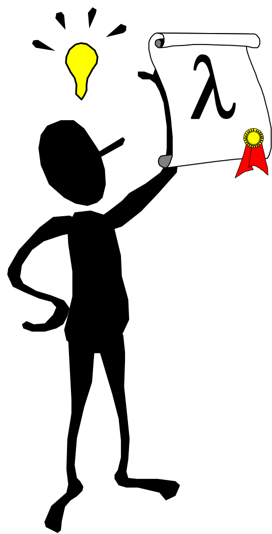


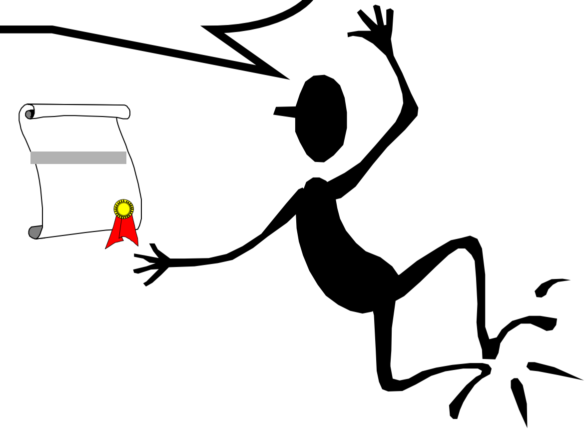


Code producer

Host

A Logical Approach





Code producer

Host

The Code Safety Problem

The Code Safety Problem

00111101
10010011

Native Compiler

Code

"Good" Java program

CPU

"Trusted Computing Base" (TCB)

The Code Safety Problem

Proving vs. Checking

Ⓐ

Ⓑ

rlrrllrrllrlrlrllrlrrllrrll…

Basic Concept of PCC

Proof Checker

CPU

00111101
10010011

Proof

Proving Compiler

Code

"Good" Java program

First Proof-Carrying Code Systems

[Necula, Lee 1996]

Proof-Carrying Code Issues

#1: How large is the trusted computing base?

- Machine (hardware) specification and semantics
- Safety policy
- Proof checker (and VC generator)
- The Logic itself

#2: How big are the proofs?

#3: How long does it take to produce/check the proofs?

Proof-Carrying Code Issues

#1: How large is the trusted computing base?

$$\frac{\begin{array}{ll} \Delta;\Psi;\Gamma & \vdash o_1 : \mathtt{int} \\ \Delta;\Psi;\Gamma & \vdash o_2 : \mathtt{set}_{=}(B) \\ \Delta;\Psi;\Gamma & \vdash \mathtt{rco}(r) : \tau_1 \vee \tau_2 \\ \Delta & \vdash \tau_1 \vee \tau_2 : TW \\ \Delta;\Psi;\Gamma\{r{:}\tau_1\} & \vdash o_1 : \tau_1' \\ \Delta;\Psi;\Gamma\{r{:}\tau_2\} & \vdash o_1 : \tau_2' \\ \Delta & \vdash \tau_1' \wedge \tau_{unsat}^{\kappa,B} \leq \mathtt{void} \\ \Delta & \vdash \tau_2' \wedge \tau_{sat}^{\kappa,B} \leq \mathtt{void} \\ \Delta;\Psi;\Gamma & \vdash o_3 : (\Gamma\{r{:}\tau_1\}) \rightarrow 0 \\ \Delta;\Psi;\Gamma\{r{:}\tau_2\} & \vdash I \end{array}}{\Delta;\Psi;\Gamma \vdash \mathtt{cmpjcc}\ o_1, o_2, \kappa, o_3; I}$$

Foundational Proof-Carrying Code

(as opposed to type-specialized "conventional" PCC)

- Use a completely foundational mathematical logic with no (or very few) built-in axioms.
- Safety policy, machine state and semantics, and safety proof built from the foundations of mathematical logic

√ Flexible

√ Secure (much smaller TCB)

☹ Complex?

FPCC System

[Appel, Felty 2000; Hamid et al. 2002]

Summary: Proof-Carrying Code

- Representation of logic/proofs
- Burden of proving safety on the code producer
- Consumer doesn't care how proofs constructed
- “Tamperproof”
- No cryptography/trusted third parties needed
- No effect on execution time after checking

- In practice? Lots of work remaining . . .

Thank you…

Nadeem Abdul Hamid

Yale University

`http://flint.cs.yale.edu`

nadeem@acm.org