INFO-231: Introduction to Mathematical Foundations of Security

Yan Huang

yh33@indiana.edu

Yan Huang

Research Interests:

- Security
- Algorithms
- Functional Programming Systems
- Cryptography

I am looking for *motivated undergraduate researchers*.

Course Administrivia

- Web site: http://homes.soic.indiana.edu/yh33/Teaching/I231-2016/syllabus.html
- TA: Ruiyu Zhu (zhu52@indiana.edu)

  Office Hours: Tuesday 1-2. GA 1st floor Lobby

- Textbooks: (both from Cambridge University Press)
  - [required] *Programming in Haskell*, Graham Hutton
  - [required] *A Cryptography Primer: Secrets and Promises*, Philip Klein
  - [recommended] *An Introduction to Mathematical Cryptography*, J. Hoffstein, J. Pipher, J. H. Silverman

Goals of this course

- Stimulate your interests in
  - Mathematics
  - Computer programming

- Some useful Math ideas
  - Prepare for later courses
  - Benefit your future career

Components of this course

Haskell programming

Algebra

Probability

Computational Complexity

Applications

Grades

Home work	40%
Quiz	20%
Final	40%

- ✓ Every homework assignment counts.
- ✓ No late homework will be accepted.
- ✓ Final grades are curved at the end of the semester.

Homework Policy

- You can discuss the problems with other students in the class, but everyone should type up the answers ***independently***.
- On your submitted paper
  - Credit who you have obtained help from
  - Write down who you have offered help to
- **Plagiarism** will always be reported and cause a failure of this course.

More policies

- Quizzes
  - Closed book, Closed notes
  - Can happen during any lecture
  - Zero point on quizzes in lectures of your absence
  - Three worst scores automatically dropped (e.g., due to missing attendance)
  - Class attendance is required *unless* you demonstrate to me that you mastered the lecture contents in advance. Must obtain permission to skip lectures.
- Final
  - Open book, take home
  - No collaboration
  - Must type up and submit electronically

How to get an A^+?

- Factorize the following number

```
1897210330998318542207007978428413548924709284842264628618381847324 95
8868359441695218259424097501740146491484482964405747209135261379874 37
4733577732309055538922373030847840111688189474510815793790974478228 81
6674328829043793821927657853344846260929644917245676138956585736358 23
4403207041644454301546146112289648218961079659268383833898994071602 91
0097071652037284416931910543644807043465629930295456867862439420227 22
5473241635983110767156374281981664270363281334019108602180065530013 25
9910552594009900649044992884447518970458977007265551419983110626457 69
9364917320085775518118977975228002508996327580943472240805266199 3
```

- Finish your assignments reasonably well and demonstrate your ability of ***proactive learning***
  - Study relevant materials not covered/required in class
  - Implement challenging stuff
  - Solve optional problems

Do talk to me in advance to settle down your specific plans

Environment Setup

Bring your laptop to class

- Quizzes
- Try out ideas on the fly.

Madoko

- https://www.madoko.net/
  - Connects well with Dropbox, Github, Onedrive etc.
- Detailed reference manual: http://research.microsoft.com/en-us/um/people/daan/madoko/doc/reference.html
- You are *required* to type up your assignments using either Madoko or LaTeX.

Daan Leijen, creator of Madoko.

Madoko

- Italic

  important => *important*

- Boldface

  important => **important**

- Inline math

  $ f(x) = x^2 + 1 $

- Displayed math

  ~ Equation { #eqn-label }
  W = F \cdot s
  ~

Madoko

- Superscripts (^) and Subscripts (~)
  - E.g., `Black_pit_, Ball~sky~`
- Strike out (~~, two tildes)
  - E.g., `There is a ~~strike out~~ here.`
- Links
  - E.g., `[Google](http://www.google.com).`
- Images

```
![bfly]
[bfly]: images/butterfly-200.png "A Monarch" { width:
100px }
```

Madoko Blocks

- Block for Displayed math

```
~ Equation
W = F \cdot s
~
```

- Equivalently

```
~ Begin Equation
  W = F \cdot s
~
```

- Nested blocks

```
~ Equation
  F=ma
~~ Equation
    An nested equation distinguished by double tildes
~~
~
```

Madoko

- $...$ marks the region where LaTeX *math-mode* applies
- LaTeX symbol lookup: http://detexify.kirelabs.org/classify.html

$\frac{1}{n}$	`\frac{1}{n}`
x_1	`x_1`
$\lim$	`\lim`
$\mod$	`\mod`
$\rightarrow$	`\rightarrow`
∞	`\infty`

Madoko

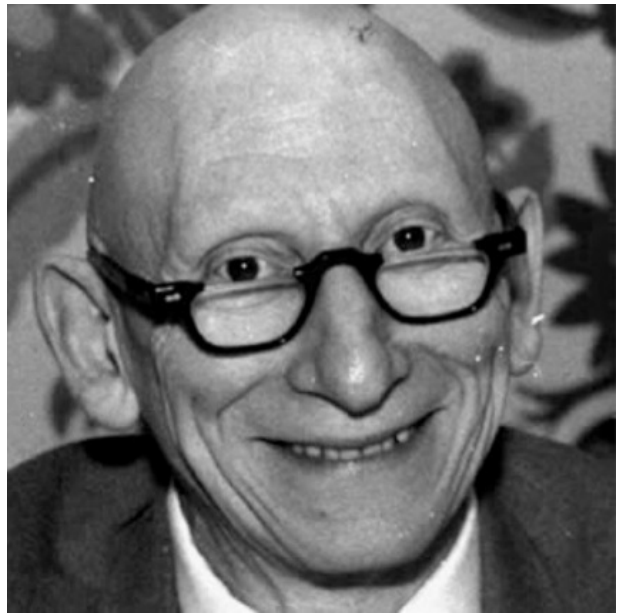
- Embedding program code

```haskell
main = print “Hello World!”
```

Why Haskell?

- Present math ideas
  - Precise
  - Succinct
  - Easy to experiment
- A bonus skill to your adventurous future
  - Functional programming: the basic method of computation is **application of functions to arguments**

Why Haskell?



A language that doesn't affect the way you think about programming is not worth knowing.

A good programming language is a conceptual universe for thinking about programming.

-- Alan Perlis
Professor of Yale
The first Turing Award Laureate

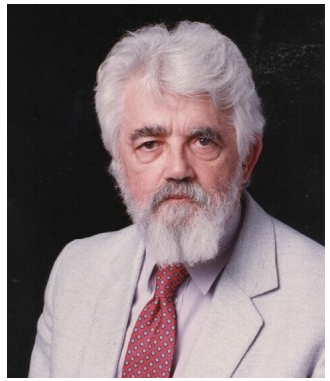
Historical Background

1930s:

Alonzo Church develops the lambda calculus, a simple but powerful theory of functions.

Historical Background

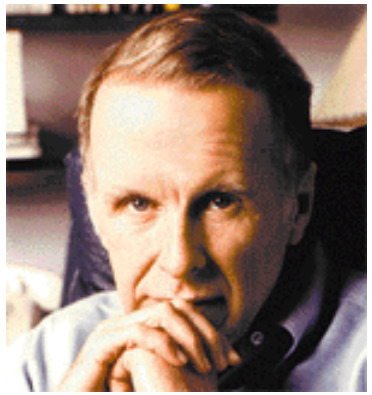
1950s:



John McCarthy develops Lisp, the first functional language, with some influences from the lambda calculus, but retaining variable assignments.

Historical Background

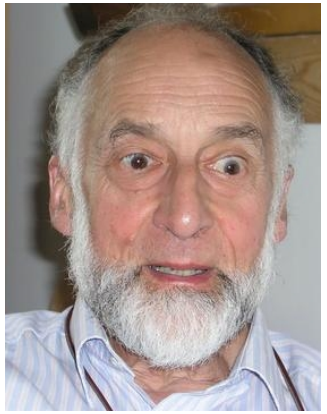
1970s:



John Backus develops FP, a functional language that emphasizes *higher-order functions* and *reasoning about programs*.

Historical Background

1970s:



Robin Milner and others develop ML, the first modern functional language, which introduced *type inference* and *polymorphic types*.

Historical Background

1987:

An international committee of researchers initiates the development of Haskell, a standard lazy functional language.

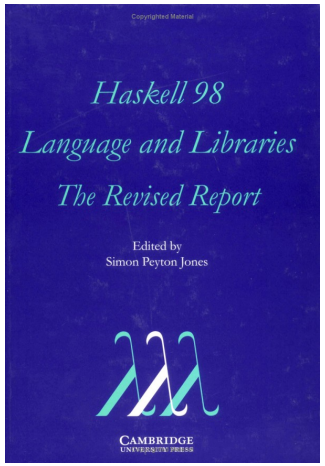
Historical Background

1990s:

Phil Wadler and others develop *type classes* and *monads*, two of the main innovations of Haskell.

Historical Background

2003:



The committee publishes the Haskell Report, defining a stable version of the language; an updated version was published in 2010.

Historical Background

2010-date:

Standard distribution, library support, new language features, development tools, use in industry, influence on other languages, etc.

Example Practical Uses

- Haxl — Facebook's anti-spam program
- Cryptol — A language and toolchain for developing and verifying cryptography algorithms
- seL4 — The first formally verified microkernel

…

Example Haskell Code

- Summing the integers 1 to 10 in Java:

```
int total = 0;
 for (int i = 1; i <= 10; i++)
   total = total + i;
```

The method of computation is *variable assignment*.

- In Haskell, this is simply a one-liner

```
sum [1..10]
```

Installing Haskell

GLASGOW HASKELL COMPILER (GHC)

- Freely available
  https://www.haskell.org/platform/
- A leading implementation of Haskell comprising a compiler `ghc` and an interpreter `ghci`
- The interactive nature of the interpreter makes it well-suited for teaching and prototyping

Starting GHCi

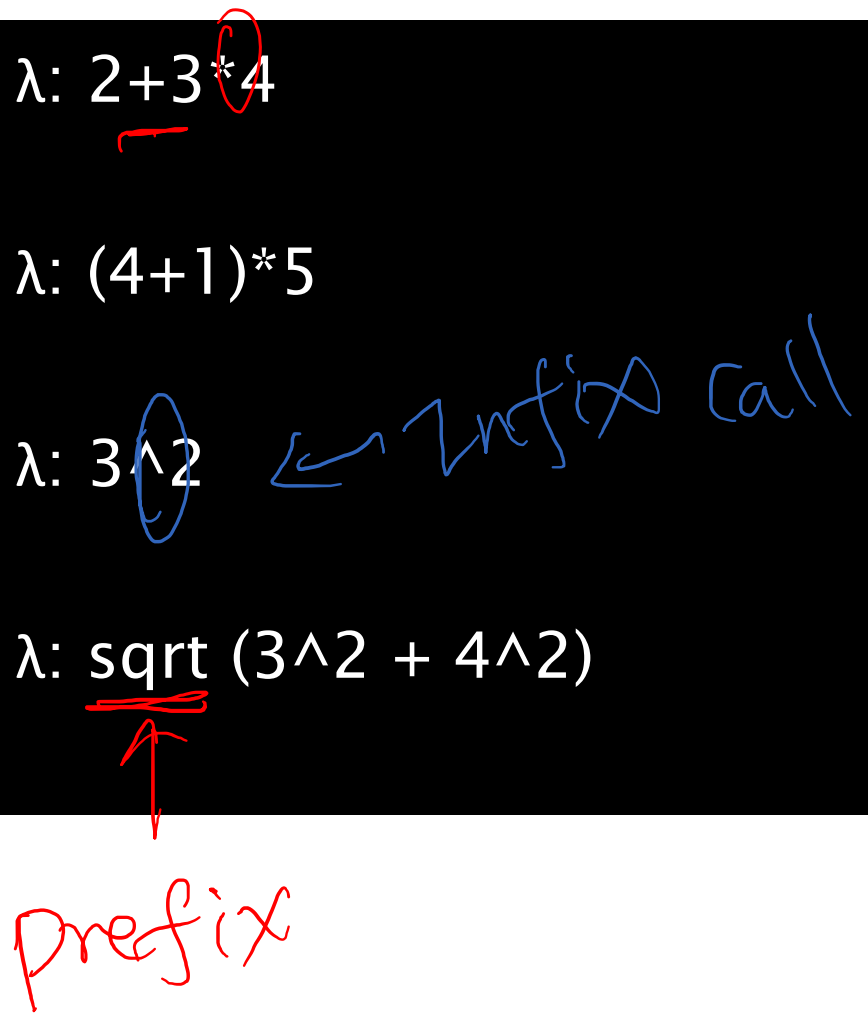
REPL

```
$ ghci
GHCi, version X: http://www.haskell.org/ghc/  :? for help
Prelude λ:
```

"Prelude λ:" prompts for Haskell expressions to evaluate.

GHCi as a desktop calculator

```
Prelude λ: 2+3*4
14
Prelude λ: (4+1)*5
25
Prelude λ: 3^2
9
Prelude λ: sqrt (3^2 + 4^2)
5.0
```



The Standard Prelude

Haskell comes with a large number of standard library functions. E.g., the library also provides many useful functions on lists.

- Select the first element of a list:

```
Prelude λ: head [1,2,3,4,5]
1
```

- Remove the first element from a list:

```
Prelude λ: tail [1,2,3,4,5]
[2,3,4,5]
```

- Select the nth element of a list: (list index starts from **0**)

```
Prelude λ: [1,2,3,4,5] !! 2
3
```

- Select the first n elements of a list:

```
Prelude λ: take 3 [1,2,3,4,5]
[1,2,3]
```

- Remove the first n elements from a list:

```
Prelude λ: drop 3 [1,2,3,4,5]
[4,5]
```

- Calculate the length of a list:

```
Prelude λ: length [1,2,3,4,5]
5
```

- Calculate the sum of a list of numbers:

```
Prelude λ: sum [1,2,3,4,5]
15
```

- Calculate the product of a list of numbers:

```
Prelude λ: product [1,2,3,4,5]
120
```

- Append two lists:

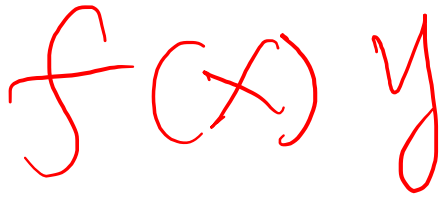
```
Prelude λ: [1,2,3] ++ [4,5]
[1,2,3,4,5]
```

- Reverse a list:

```
Prelude λ: reverse [1,2,3,4,5]
[5,4,3,2,1]
```

Function Application

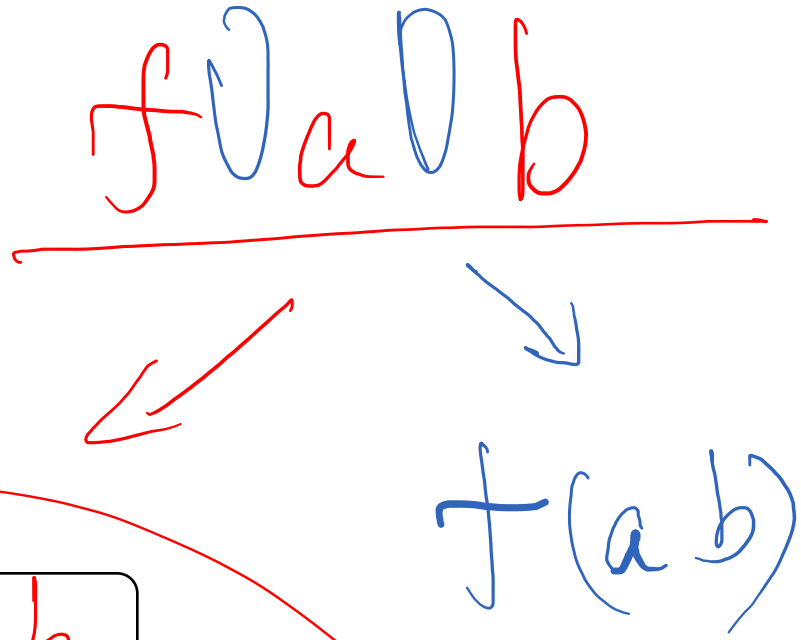
In mathematics, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space.




```
f(a,b) + c d
```

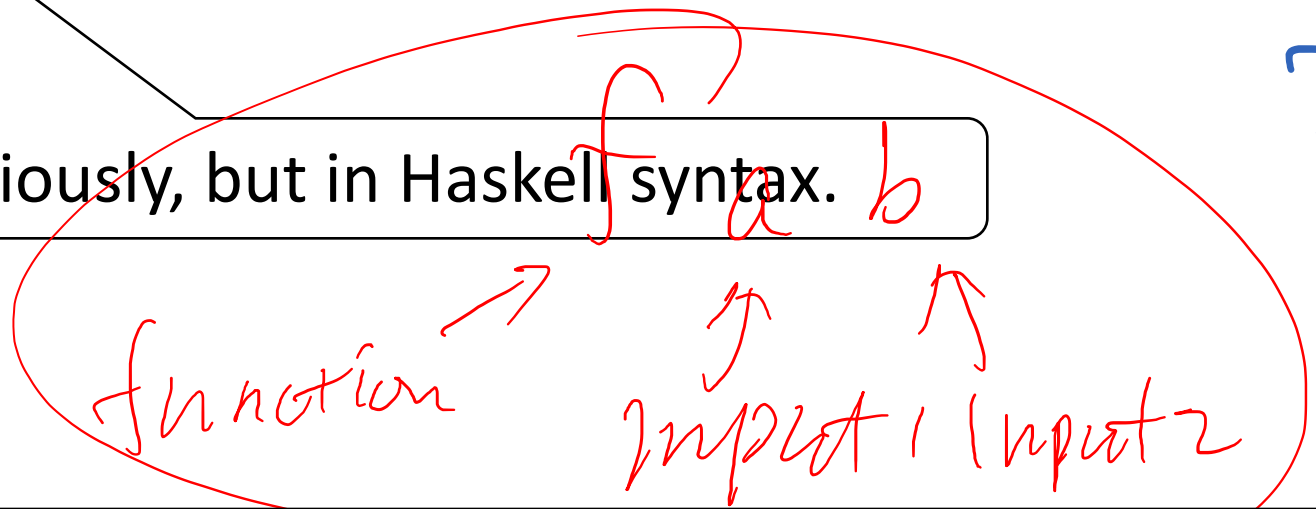
Apply the function f to a and b, and add the result to the product of c and d.

In Haskell, function application is denoted using space, and multiplication is denoted using *.



```
f a b + c*d
```

As previously, but in Haskell syntax.



Moreover, function application is assumed to have higher priority than all other operators.

```
f a + b
```

Means (f a) + b, rather than f (a + b).

Examples

Mathematics	Haskell
f(x)	f x
f(x,y)	f x y
f(g(x))	f (g x)
f(x,g(y))	f x (g y)
f(x)g(y)	f x * g y
f(-1)	f (-1)

f(x, y)

f - 1

will be treated as subtraction

Useful GHCi Commands

Command	Meaning
:load *name*	load script *name*
:reload	reload current script
:set editor *name*	set editor to *name*
:edit *name*	edit script *name*
:edit	edit current script
:type *expr*	show type of *expr*
:?	show all commands
:quit	quit GHCi

:type 5 ::Int

Charge

- Haskell
  - Install Haskell Platform on your computer.
  - Try out the GHCi evaluations covered in this lecture.
- Madoko
  - Read through Madoko's reference manual
  - Try out Madoko tricks as you read the manual.
- Homework 0 announced
  - **Start early!**
  - Submit through Canvas