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
- [required] Programming in Haskell, Graham Hutton
- [required] A Cryptography Primer: Secrets and Promises, Philip Klein
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
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Home work</td>
<td>40%</td>
</tr>
<tr>
<td>Quiz</td>
<td>20%</td>
</tr>
<tr>
<td>Final</td>
<td>40%</td>
</tr>
</tbody>
</table>

✓ 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

```
189721033099831884220700797842841354892470928484226462861838184732495
886835944169521825942409750174014649148448296440574720913526137987437
473357773230905553892237303084784011168818947451081579379097447822881
667432882904379382192765785334484626092964491724567613895658573635823
440320704164445430154614611228964821896107965926838383389899407160291
009707165203728441693191054364480704346562993029545686786243942022722
547324163598311076715637428198166427036328133401910860218006553001325
99105525940099006490449288444751897045897700726555141998311062645769
93649173200857755181189779752280025089963275809434722408052661993
```

• 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/](https://www.madoko.net/) - Connects well with Dropbox, Github, Onedrive etc.
- 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
  \[ W = F \cdot s \]
  ~

• Nested blocks
  ~ Equation
  \[ F = ma \]
  ~~ Equation
  An nested equation distinguished by double tildes
  ~~
  ~

• Equivalently
  ~ Begin Equation
  \[ W = F \cdot s \]
  ~
Madoko

- \dots marks the region where LaTeX *math-mode* applies

### LaTeX symbol lookup:

- \( \frac{1}{n} \)
- \( x_1 \)
- \( \lim \)
- \( \mod \)
- \( \rightarrow \)
- \( \infty \)

<table>
<thead>
<tr>
<th>( \frac{1}{n} )</th>
<th>( \frac{1}{n} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( x_1 )</td>
</tr>
<tr>
<td>( \lim )</td>
<td>( \lim )</td>
</tr>
<tr>
<td>( \mod )</td>
<td>( \mod )</td>
</tr>
<tr>
<td>( \rightarrow )</td>
<td>( \rightarrow )</td>
</tr>
<tr>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>

[LaTeX symbol lookup: http://detexify.kirelabs.org/classify.html](http://detexify.kirelabs.org/classify.html)
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.
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:

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

The method of computation is \textit{variable assignment}.

• In Haskell, this is simply a one-liner

```haskell
sum [1..10]
```
Installing Haskell

GLASGOW HASKELL COMPILER (GHC)

• Freely available
  [https://www.haskell.org/platform/](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

```
$ 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
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:

  \[
  \text{Prelude } \lambda: \text{drop 3 [1,2,3,4,5]} \\
  [4,5]
  \]

- Calculate the length of a list:

  \[
  \text{Prelude } \lambda: \text{length [1,2,3,4,5]} \\
  5
  \]

- Calculate the sum of a list of numbers:

  \[
  \text{Prelude } \lambda: \text{sum [1,2,3,4,5]} \\
  15
  \]
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.

\[ f(a, b) + c \cdot d \]

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 *.

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

<table>
<thead>
<tr>
<th>Mathematics</th>
<th>Haskell</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(x)</td>
<td>f x</td>
</tr>
<tr>
<td>f(x,y)</td>
<td>f x y</td>
</tr>
<tr>
<td>f(g(x))</td>
<td>f (g x)</td>
</tr>
<tr>
<td>f(x,g(y))</td>
<td>f x (g y)</td>
</tr>
<tr>
<td>f(x)g(y)</td>
<td>f x * g y</td>
</tr>
<tr>
<td>f(-1)</td>
<td>f (-1)</td>
</tr>
</tbody>
</table>

Note: F(x,y) will be treated as subtraction.
# Useful GHCi Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>:load <em>name</em></td>
<td>load script <em>name</em></td>
</tr>
<tr>
<td>:reload</td>
<td>reload current script</td>
</tr>
<tr>
<td>:set editor <em>name</em></td>
<td>set editor to <em>name</em></td>
</tr>
<tr>
<td>:edit <em>name</em></td>
<td>edit script <em>name</em></td>
</tr>
<tr>
<td>:edit</td>
<td>edit current script</td>
</tr>
<tr>
<td>:type <em>expr</em></td>
<td>show type of <em>expr</em></td>
</tr>
<tr>
<td>:?</td>
<td>show all commands</td>
</tr>
<tr>
<td>:quit</td>
<td>quit GHCi</td>
</tr>
</tbody>
</table>
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