Welcome!

Simone Campanoni
simonec@eecs.northwestern.edu
Who we are

Simone Campanoni
simonec@eecs.northwestern.edu

Enrico A. Deiana
enricodeiana2020@u.northwestern.edu
What we are going to do

• Teach you **code analysis and transformation**

• What they do
• What they could do
• What they can’t do
Who you are (or will be)

• An engineer

• A C++ developer
  (you don’t have to be an incredible coder)

• An enthusiastic learner

Compiler expert is not mentioned ;}
Outline of today’s CAT

• Structure of the course

• CAT and compilers

• CAT and computer architecture

• CAT and programming language
CAT in a nutshell

• **About:** understanding and transforming code automatically
• EECS 396/496
• Satisfy the system depth for CS major
• Tuesday/Thursday 2:00pm – 3:20pm at LR2 Tech (here ;))

• Simone’s office hours: Friday 2:00pm – 4:00pm
  • But feel free to stop by at my office (2.217@Ford) any time
• Enrico’s office hours: Monday 2:00pm – 3:00 pm, 2.227@Ford

• CAT is on Canvas
  • Materials/Calendar/Assignments/Grades on Canvas
  • You’ll upload your assignments on Canvas
CAT materials

- Modern compiler implementation
- Slides and assigned papers
- LLVM documentation

http://llvm.org
The CAT structure

Week

Tuesday  Thursday

Homework

Today  12/8
The CAT grading

• Homework: 100 points
  • 10 points per assignment
  • The first 2 assignments are trivial

• Extra points
  • Extra homework
  • Answering (correctly) special questions (I will emphasize them) during lectures
  • Best student so far: **114 points**!

<table>
<thead>
<tr>
<th>Grade</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>95 – 100</td>
</tr>
<tr>
<td>A -</td>
<td>90 – 94</td>
</tr>
<tr>
<td>B +</td>
<td>80 – 89</td>
</tr>
<tr>
<td>B</td>
<td>70 – 79</td>
</tr>
<tr>
<td>B -</td>
<td>61 – 69</td>
</tr>
<tr>
<td>C +</td>
<td>57 – 60</td>
</tr>
<tr>
<td>C</td>
<td>50 – 56</td>
</tr>
<tr>
<td>D</td>
<td>25 – 49</td>
</tr>
<tr>
<td>F</td>
<td>0 – 24</td>
</tr>
</tbody>
</table>
Rules for homework

• No copying of code is allowed

• Tool, infrastructure help is allowed
  • First try it on your own
    (google and tool documentation are your friends)

• Avoid plagiarism
  www.northwestern.edu/provost/policies/academic-integrity/how-to-avoid-plagiarism.html

• If you don’t know, please ask
  simonec@eecs.northwestern.edu
Summary

• My duties
  • Teach you code analysis and transformation
  • And how to implement them in a production compiler (LLVM)

• Your duties
  • Learn code analysis and transformation
  • Implement a few of them in LLVM
    • Write code
    • Test your code
    • Then, think much harder about how to actually test your code
    • (Sometimes) Answer my questions about your code
Structure & flexibility

• CAT is structured w/ topics

• Best way to learn is to be excited about a topic

• Interested in something?

   Speak

   I’ll do my best to include your topic on the fly
Week 1

Today
- Welcome/Structure
- Compiler/CAT

Thursday
LLVM

Today
- Welcome/Structure
- Compiler/CAT

12/8
The role of compilers

If there is no coffee, if I still have work to do, I'll keep working, I'll go to the coffee shop.
Example of CAT

varX = 5
...
...
...
...
print varX
...

What will it print?
Example of CAT

```python
varX = 5
...  
...  
...  
What will it print?  
print 5  
...  
print varX
```
Example of CAT

\[
\begin{align*}
\text{varX} &= 5 \\
\text{...} & \quad \text{...} \\
\text{...} & \quad \text{...} \\
\text{...} & \quad \text{...} \\
\text{...} & \quad \text{...} \\
\text{print 5} & \quad \text{print varX} \\
\text{...} & \quad \text{...}
\end{align*}
\]

Is it worth transforming?

Analysis

Property

Transformation

Transformed code

Code
Designing CATs

• Choose a goal
  • Performance, energy, identifying bugs, discovering code properties

• Design automatic analysis to obtain the required information

• Occasionally design the code transformation
Use of CATs

- Compilers
  - Increase performance
  - Decrease energy consumption
  - Code generation

- Developing tools (e.g., VIM, EMACS)
  - Understanding code (e.g., scopes, variables)

- Computer architecture
Structure of a compiler

Character stream (Source code)

Lexical analysis

Tokens

Syntactic & semantic analysis

AST

```
int main()
{
    printf("Hello World\n");
    return 0;
}
```
Structure of a compiler

Character stream (Source code)

Lexical analysis

Tokens

Syntactic & semantic analysis

AST

```
int main ...
```

```
INT SPACE STRING SPACE ...
```

```
Function signature
```

```
Return type
```

```
INT
```

```
Function name
```

```
STRING
```
Structure of a compiler

Syntactic & semantic analysis

AST

IR code generation

Function signature

Return type

Function name

INT

STRING

; Function Attrs: nounwind uwtable
define int @main() {

Structure of a compiler

Character stream (Source code) → Front-end → IR → Middle-end → IR → Back-end → Machine code

```
int main(...)
```

---

**EECS 322: Compiler Construction**

Code analysis and transformation

```
; FunctionAttrs: nounwind uwtable
define int @main() {
```

---

**EECS 322: Compiler Construction**

```
010101110101010101
```
Structure of a compiler

Character stream (Source code) → Front-end → Middle-end → Back-end → Machine code

Front-end

Middle-end

Back-end

Machine code
Structure of a compiler

- Front-end
  - IR
  - Middle-end
    - IR
    - Back-end
      - Machine code

- C
  - Java
  - Front-end
    - Middle-end
    - Back-end
      - Machine code
Structure of a compiler

C
Front-end
IR
Middle-end
IR
Back-end
Machine code

Java
Front-end
Middle-end
Back-end
Machine code
Structure of a compiler

- Front-end
  - Middle-end
  - Back-end
    - Machine code

- Java

- C
  - IR
  - Front-end
    - Middle-end
      - Back-end
        - Machine code

- M2
Structure of a compiler

Front-end → IR → Middle-end → IR → Back-end

C → Front-end → IR → Middle-end → IR → Back-end → Machine code

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Machine code → BE → M2

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Java → Front-end → IR → Middle-end → IR → Back-end → Machine code

Machine code → BE → M2
Structure of a compiler

Front-end 1 → IR → Middle-end → IR → Back-end A

Front-end 2 → IR → Middle-end → IR → Back-end B
Multiple IRs

• Abstract Syntax Tree

• Register-based representation (three-address code)
  \[ R1 = R2 + R3 \]

• Stack-based representation
  \texttt{push 5; push 3; add; pop ;}

IR needs to be easy
1) to produce
2) to translate into machine code
3) to transform/optimise
Example of LLVM IR

define i32 @main(i32 %argc, i8** %argv) {
entry:
  %add = add i32 %argc, 1
  ret i32 %add
}
Multiple IRs used together

L1

Static compiler

IR1

Dynamic compiler FE

IR2

Dynamic compiler BE

Machine code
Multiple IRs used together

Java

Java compiler

Java bytecode

Java VM FE

IR2

Java VM BE

Machine code
CATs that we’ll focus on

• Semantics-preserving transformations
  • Correctness guaranteed

• Goal: performance

• Automatic

• Efficient
Evolution of CATs (hardware point of view)

- Simple hardware (few resources), simple CATs
Evolution of CATs (hardware point of view)

- Simple hardware (few resources), simple CATs
- More hardware resources available to compilers
- Opportunities to improve programs
- Challenging CATs

Compilers/CATs are developed in the processor-design stage!
Evolution of CATs (hardware point of view) (2)

1960 - ?: Complex instruction set computing (CISC)

1980 - ?: Reduced instruction set computer (RISC)
Evolution of CATs (hardware point of view) (3)

Superscalar

Inst 1
Inst 2
Inst 3
Inst 4
Inst 5
Inst 6
Inst 7
Inst 8

CATs

Very long instruction word (VLIW)

Inst 1
Inst 4
Inst 7
Inst 8

Inst 2
Inst 5
Inst 3
Inst 6
Evolution of CATs (PL point of view)

• First electronic computers appeared in the ’40s
• They were programmed in machine language

• Low level operations only
  • Move data from one location to another
  • Add the contexts of two registers
  • Compare two values

• Programming: slow, tedious, and error prone
Evolution of CATs (PL point of view)

• Low level programming language, simple CATs
  • Not very productive

• More abstraction in programming language, more work for CATs to reduce their performance overhead
  • Macros -> Fortran, Cobol, Lisp -> C, C++, Java, C#, Python, PHP, SQL, ...

• CATs enable new programming languages
Evolution of CATs (PL point of view)

• Abstractions are great for productivity

• CATs remove their overhead

• But abstractions must be carefully evaluated considering CATs

• A simple abstraction in PL can generate challenges for CATs
  • CATs need to be understood
Evolution of CATs (PL point of view)(2)

PL without procedures

void main (){  
    Int v1,v2;  
    v1 = 1;  
    v2 = 2;  
    ...  
}
Evolution of CATs (PL point of view)(3)

Let’s add procedures to our PL

• Call-by-Value

```c
void proc1 (int a){...
proc1(myVar1);
```

• Call-by-Reference

```c
void proc1 (int a){...
proc1(myVar1);
```
Evolution of CATs (PL point of view)(2)

```c
void myProc (int *v1, int *v2){
    (*v1) = 1;
    (*v2) = 2;
}
```

What's the problem for CATs? ... if v1 and v2 alias ...

Understanding if pointers alias: pointer alias analysis

This is one of the most challenging problem in CATs
Conclusion

• CATs used for multiple goals
  • Enable PLs
  • Enable hardware features

• CATs are effected by
  • Their input language
  • The target hardware

• When you design a PL or a new hardware platform, you need to understand what CATs can and can’t do
  • Some cant’s become can thanks to research on CATs
Ideal CATs

• Proved to be correct

• Improve performance of many important programs

• Minor compilation time

• Negligible implementation efforts
As Linus Torvalds says ...

*Talk is cheap. Show me the code.*

Demo time