Lecture 2

COMPILER DESIGN

Administrivia

- **Web site:** https://people.inf.ethz.ch/suz/teaching/252-0210.html
- **Moodle:** https://moodle-app2.let.ethz.ch/user/index.php?id=13479
- **E-mail for teaching staff:** cd-tas-f20@lists.inf.ethz.ch
  - Please use only for private questions

- Please check course Moodle and Web sites regularly
  - Announcements
  - Homework assignments
  - Exercise sessions
  - Q/A and discussions

The Compiler Project

- Course projects (**50%** of grades)
  - (**5%**) HW1: Ocaml programming
  - (**9%**) HW2: X86lite interpreter
  - (**9%**) HW3: LLVMlite compiler
  - (**9%**) HW4: Lexing, parsing, simple compilation
  - (**9%**) HW5: Higher-level features
  - (**9%**) HW6: Analysis and optimizations

- Goal: Build a *complete compiler* from a high-level, type-safe language to x86 assembly

HW1: Hellocaml

- Homework 1 is now available on the course Moodle site
  - Individual project – no groups (**5% of overall grade**)
  - Topic: OCaml programming, an introduction to interpreters
  - **If you haven't yet, please start learning OCaml now!**

- OCaml head start
  - Run "ocaml" from the command line to invoke the top-level loop
  - Run "ocamlbuild main.native" to run the compiler

- We recommend using
  - Emacs/Vim + merlin
  - (less recommended: Eclipse with the OcaIDE plugin)

  - More information on the tool chain will be on course Moodle

How to represent programs as data structures.
How to write programs that process programs.

INTERPRETERS

Factorial: Everyone's Favorite Function

- Consider this implementation of factorial in a hypothetical programming language:

```
X = 6;
ANS = 1;
whileNZ (x) {
    ANS = ANS * X;
    X = X + -1;
}
```

- We need to describe the constructs of this hypothetical language
  - ***Syntax***: which sequences of characters count as a legal "program"?
  - ***Semantics***: what is the meaning (behavior) of a legal "program"?

Grammar for a Simple Language

```
<exp>  ::=
       |     <X>
       |     <exp> + <exp>
       |     <exp> * <exp>
       |     <exp> < <exp>
       |     <integer constant>
       |     (<exp>)

<cmd>  ::=
       |     skip
       |     <X> = <exp>
       |     ifNZ <exp> { <cmd> } else { <cmd> }
       |     whileNZ <exp> { <cmd> }
       |     <cmd>; <cmd>
```

- Concrete syntax (grammar) for a simple imperative language
  - Written in "Backus-Naur form"
  - <*exp*> and <*cmd*> are *nonterminals*
  - '::=' , '|' , and <…> symbols are part of the *meta* language
  - keywords, like '`skip`' and '`ifNZ`' and symbols, like '`{`' and '`+`' are part of the *object* language

- Need to represent the *abstract syntax* (i.e. hide the irrelevant of the concrete syntax)
- Implement the *operational semantics* (i.e. define the behavior, or meaning, of the program)

OCaml Demo

simple.ml

translate.ml