Compiler Construction
Winter 2020

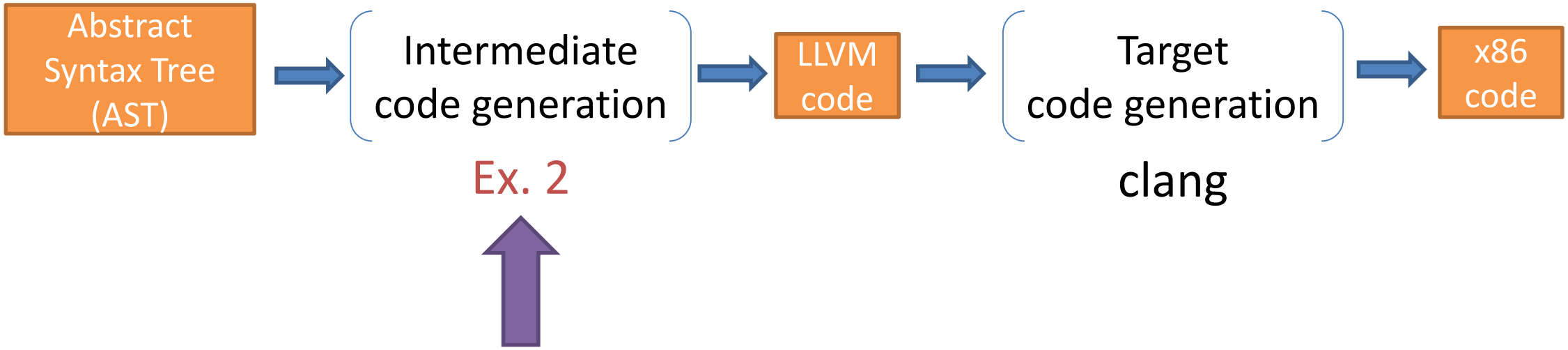
Recitation 5:
Code Generation*

* Low-level IR

Yotam Feldman

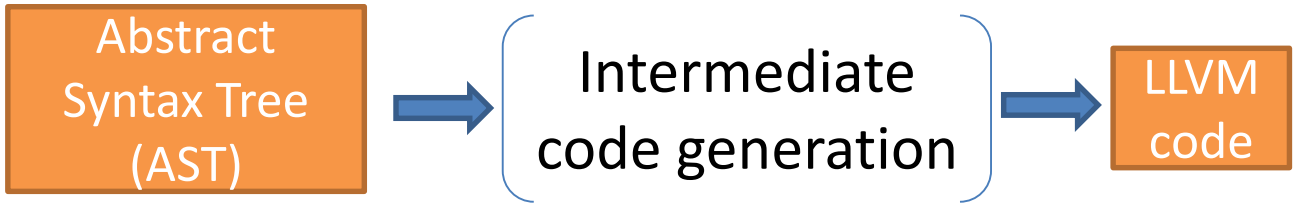
Based on materials by Yannis Smaragdakis
and slides by Guy Golan-Gueta

Code Generation



Today: compiling basic, imperative features

Code Generation



- Valid programs (ASTs) compile to an LLVM program that's
  - valid,
  - executes,
  - has the same input-output and external behavior (console output)
- Rules for valid MiniJava ASTs: https://www.cs.tau.ac.il/research/yotam.feldman/courses/wcc20/semantic.html

LLVM Recap

- Typed
- Unbounded number of SSA registers
- Stack allocation `alloca`
- Heap allocation and `bitcast`
- `load` and `store`
- Branch and conditional branch: `br`
- Array and `getelementptr`
- Basic binary operations: `add, sub`…
- Function calls: `call` and `ret`

Translation (IR Lowering)

Visitor(s) generate LLVM declarations and code

- Class declarations
- Statements
- Expressions

Local variables

Demo

- Local variables translated to stack locations
- Load & store
  - Too early to optimize!

Store & Load According to Static Type

Demo x 2

- Assume type safety!
  - Otherwise, the behavior is **undefined**
- Use symbol table to obtain the type from the declaration

Simple Expressions

Demo

- TR[e] = LIR translation of AST expression e
  - A sequence of IR instructions
  - Use temporary variables (IR registers) to store intermediate values during translation

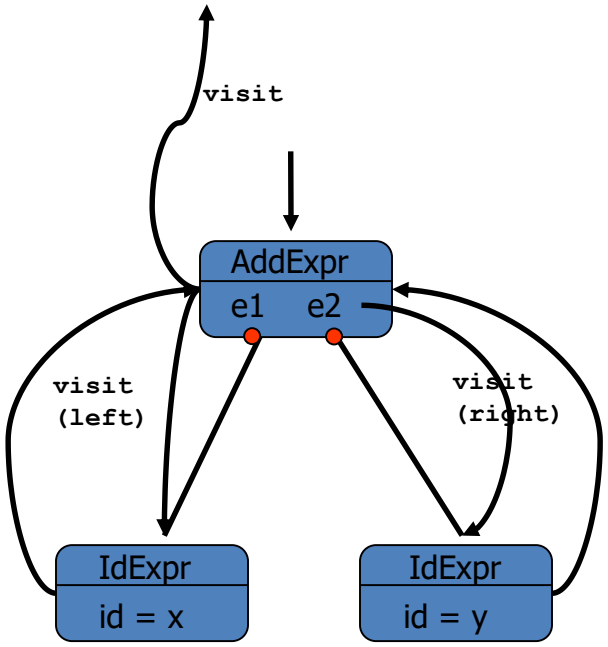
Compound Expressions

Demo

- SSA, need to allocate fresh registers
- Order of evaluation is important
  - Think about method calls that perform mutations

Compound Expressions: Example

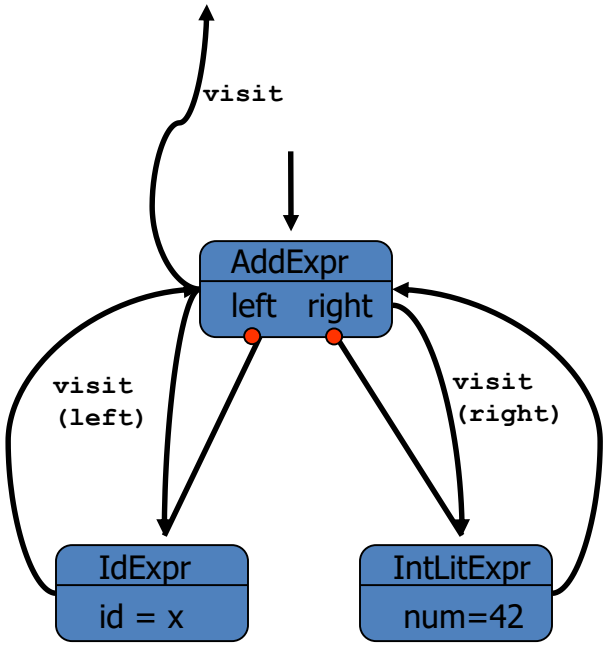
`TR[x + y]`




```
%_0 = load i32, i32* %x
%_1 = load i32, i32* %y
%_2 = add i32 %_0, %_1
```

Translating expressions – example

`TR[x + 42]`




```
%_0 = load i32, i32* %x

(%_1 = i32 42 --- invalid)

%_2 = add i32 %_0, 42
```

Translating Statement Blocks

`TR[{ s1; s2; … ; sN }]`

TR[s1]

TR[s2]

TR[s3]

…

TR[sN]

Translating If-Then-Else

Demo

- Conditional branch
- Need to generate code evaluating the condition

Translating Short-Circuit And

Demo

Generate code for

- Evaluating the first operand
- If true, continuing; otherwise skipping
- Evaluating the second operand
- Joining using the `phi` instruction

Translating While

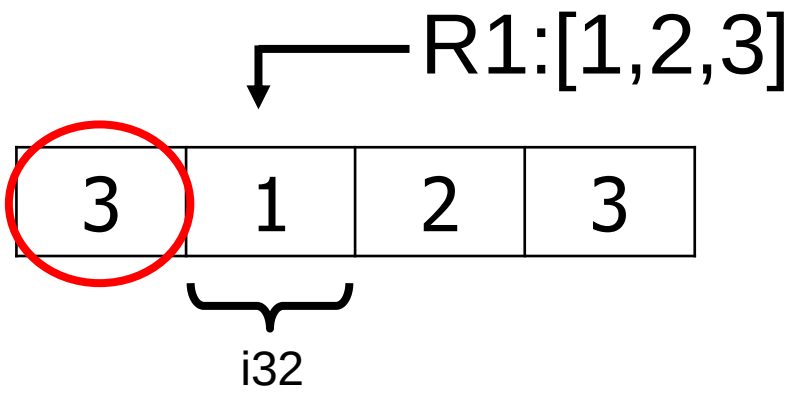
- Jump back to beginning of the loop
- Exercise ☺

Arrays

- Allocation
- Access
- Assignment
- Dynamic checks
  - “ArrayIndexOutOfBoundsException”

- Also: array length (exercise ☺)

Demo



Summary

- Local (stack) variables
- Generating code for expressions
- Control structures
- Short-circuit and
- Arrays
- Upcoming: object-oriented code generation

Exercise #2

- **Start early**
- Read the requirements carefully!
- Reference compilation examples
- Extend symbol table and class hierarchy analysis from ex1
- Assume that the program is semantically valid
  - List of rules
  - For the type in LLVM instructions, use the declared type and assume that the usage is valid
- Class fields and (instance) method calls – next week
  - From today: local variables, expressions, control flow, arrays…
- Submission instructions