Code Generation

Martin Sulzmann

Code Generation

Objective

Program in intermediate code representation.

- Produce correct output code.
- Produce efficient output code.
- Ensure efficient code generation.

Things to consider

- Must trike a balance.
- What's the *target* language?
  - Assembler code (ARM, ...).
  - C code.
  - Virtual machine (VM).

Optimizations

Target Independent (generally)

- Simplifications.
- Dead code removal.
- ...

Target Dependent (generally)

- Register allocation.
- Instruction selection (to optimally fill instruction pipeline).
- ...

Our Approach

Stack-based VM

- Control stack to manage activation records.
- Linear memory address space.
- Stack machine for computations.

Pros

- Simple code generator (and simple VM interpreter).
- Compact code.

Cons

- Requires fast underlying implementation.
- More memory references.

Stack Machine

Expression

```
(1 + 2) * 3
```

VM Code

Computations via stack. Reverse polish notation (postfix).

```
PushS 1
PushS 2
Add
PushS 3
Mult
```

Memory Interaction

Expression

```
x = y + 1
```

VM Code

```
PushToStack LocY                    -- Memory address of y
PushS 1
Add
AssignFromStack (0,LocX)            -- Relative access on stack, c
PopS                                -- Clean up stack
```

Control Flow

Program code

```
if 1 < 2 { x = 1 }
else { x = 0 }
...
```

VM Code

```
1: PushS 1
2: PushS 2
3: Gt
4: NonZero 7
5: Assign (LocX, 0)
6: Jump 8
7: Assign (LocX, 1)
8: ...
```

Procedures

Control Stack

- Separate stack to manage activation records.
- Layout as described earlier.