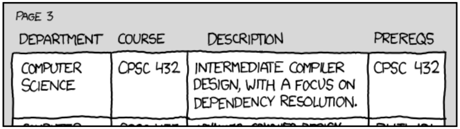
Global Value Numbering in Factor

Alex Vondrak

ajvondrak@csupomona.edu

September 1, 2011



PAGE 3

DEPARTMENT	COURSE	DESCRIPTION	PREREQS
COMPUTER SCIENCE	CPSC 432	INTERMEDIATE COMPILER DESIGN, WITH A FOCUS ON DEPENDENCY RESOLUTION.	CPSC 432

Factor

Factor (`http://factorcode.org/`)

- Started development September 2003—a baby among languages
- Stack-based
- Object-oriented
- Dynamically typed
- Extensive standard library
- High-level, yet fully compiled

Won't really have time to discuss the language in depth

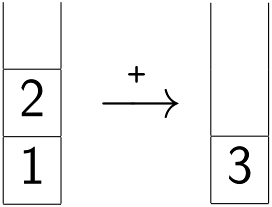
Stacks as an Evaluation Model

Example (Code)

```
1 2 +
```

Example (Execution)

```
push(1);
push(2);
y = pop();    // y = 2;
x = pop();    // x = 1;
push(x + y); // push(3);
```



Organization

Non-optimizing base compiler

- VM written in C++
- Responsible for basic runtime services
    - Garbage collection
    - Method dispatch
    - Polymorphic inline caches
    - ...
- Single pass—outputs assembly stubs for primitives

Optimizing compiler

- Written in Factor code
    - Possible by *bootstrapping*
- Optimizes in passes across two intermediate representations (IRs)
    - High-level IR (`compiler.tree`)
    - Low-level IR (`compiler.cfg`)

High-level IR

- Tree of node objects
- Very simple virtual instruction set
  - #introduce, #return
  - #push & #call
  - #renaming—#copy & #shuffle
  - #declare & #terminate
  - #branch—#if & #dispatch
  - #phi
  - #recursive, #enter-recursive, #call-recursive, #return-recursive
  - #alien-node, #alien-invoke, #alien-indirect, #alien-assembly, #alien-callback
- Input/output values of stack given unique names

High-level IR

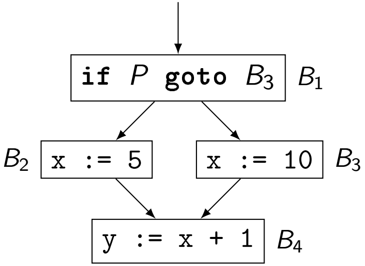
1 2 +

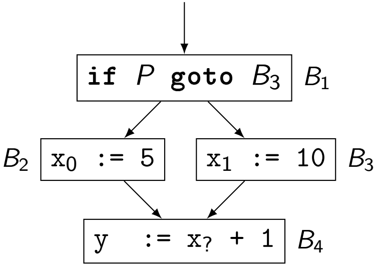
Example

```
V{
    T{ #push { literal 1 } { out-d { 6256273 } } }
    T{ #push { literal 2 } { out-d { 6256274 } } }
    T{ #call
        { word + }
        { in-d V{ 6256274 6256273 } }
        { out-d { 6256275 } }
    }
    T{ #return { in-d V{ 6256275 } } }
}
```

Low-level IR

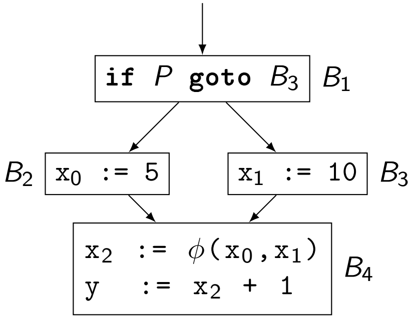
- Control flow graph (CFG)
  - Basic blocks = maximal sequence of “straight-line” code
  - Directed edges = transfer of control flow
- `insn` objects modeled closely after assembly-like instructions
- Static single assignment (SSA) form





Low-level IR

- Control flow graph (CFG)
  - Basic blocks = maximal sequence of "straight-line" code
  - Directed edges = transfer of control flow
- `insn` objects modeled closely after assembly-like instructions
- Static single assignment (SSA) form



Optimizations—High-level IR

```
: optimize-tree ( nodes -- nodes' )
  [
      analyze-recursive
      normalize
      propagate
      cleanup
      dup run-escape-analysis? [
          escape-analysis
          unbox-tuples
      ] when
      apply-identities
      compute-def-use
      remove-dead-code
      ?check
      compute-def-use
      optimize-modular-arithmetic
      finalize
  ] with-scope ;
```

Optimizations—Low-level IR

```
        : optimize-cfg ( cfg -- cfg' )
            optimize-tail-calls
            delete-useless-conditionals
            split-branches
            join-blocks
            normalize-height
            construct-ssa
            alias-analysis
→           value-numbering
            copy-propagation
            eliminate-dead-code ;
```