CS 444 - Compiler Construction **Winter 2020**

Lecture 17: March 9 , 2020

Lecturer: Ondřej Lhoták *Notes By: Harsh Mistry*

17.1 Java Reachability Continued

- Constraints Continued
  - L: return or L: return E
    - out[L]= no
  - L: $\{S_1 \dots S_k\}$
    - in$[S_1]$ = in[L]
    - in$[S_{i+1}]$ = out$[S_i]$
    - out[L] = out$[S_k]$
  - L: any other statement
    - out[L] = in[L]
  - L: body of a method/constructor
    - in[L] = maybe
- Implementing constraints
  - No cycles in constraints for this analysis
  - Just implement in/out functions directly from constraints
  - Memoize: $O(n^2) \to O(k)$

17.2 Definite assignment

- Local variables must be written before being read
  - e.g `int x; return x;` would be an error in java
  - e.g `int x; x = 5; return x;` would not be an error
  - In Joos
    - Local variable deceleration requires initializer
    - Local variable can not be in its own initializer

17.3 Code Generation - x86 code generation

- Family of languages
- Registers

  - General: `eax, ebx, ecx, edx, edx, esi, edi`
  - Stack Pointer: `esp` (Can be used for anything, but implicitly assumed to be stack pointer)
  - Frame Pointer: `ebp` (Available for anything, but again assumed to be frame pointer)

- Segment Registers
  - `cs, ds, es, ss, fs, gs`
  - To be ignored. It is a remnant of the 16-bit era allowing for more memory to be referenced with 16-bit registers
  - General idea was that a segment register would point to a 64-bit page chunk and the actual register would contain a reference within that chunk

- Instructions
  - Move (move, destination, source)

```
1 mov eax, ebx    ; eax = ebx
2 mov [eax], ebx ; *eax = ebx
3 mov eax, [esp + ebx * 2 -5] ; eax = *(esp + ebx * 2 -5)
4 mov eax, 42 ; eax = 42
5 label:
6 mov eax, label  ; eax = label_address
7 mov eax, [label]
```

  - Operators (operator,source/destination,amount)

```
1 add eax, ebx ; eax += ebx
2 sub eax, ebx ; eax -= ebx
3 imut eax, ebx ; eax *= ebx  (i means signed)
```

  - Divide (operator, divisor)

```
1 idiv ebx; eax = edx:eax / ebx ; edx = edx:eax % ebx
```

    * Set `edx` = 0 if `eax` is positive
    * Set `edx` = -1 if `eax` os negative
    * you can also just use the `cdq` instruction to do this for you