Compilers

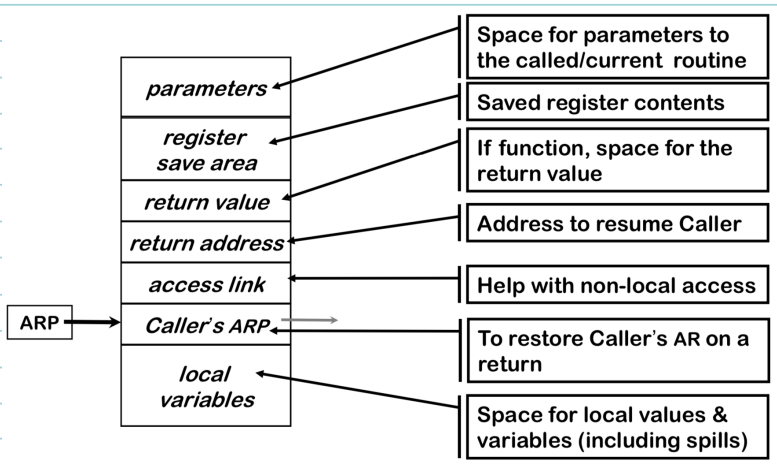
Class Notes for the second test:

1. Run-Time Environments

- What are Activation Records (AR) and what are they used for? a)
- What information does an AR capture and where are they allocated and why? b)
- Support for Object-oriented languages: class structure and inheritance. c)

a) An Activation Record (AR) is a block of memory associated with an invocation of a procedure.

b)



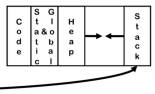
- Stack used for Activation Records (AR)
- Heap for Data (including AR) whose lifetime extends beyond activation.

Activation Records Details

Where do Activation Records live?

- If lifetime of AR matches lifetime of invocation, *AND*
- If code normally executes a "return"

⇒ Keep ARs on a stack



Yes! This stack.

- If a procedure can outlive its caller, *OR*
- If it can return an object that can reference its execution state

⇒ ARs must be kept in the Heap

- If a procedure makes no Calls

⇒ AR can be allocated statically

Efficiency prefers Static, Stack, then Heap

How does the Compiled Code finds the Variables?

- They are at known offsets from the AR pointer
- Code nesting level and AR offset within a procedure
  - **Level** specifies an ARP, offset is the constant (*later…*)

Variable-Length Data

- If AR can be extended, put it after Local Variables
- Leave a pointer at a known offset from ARP
- Otherwise, put variable-length data on the Heap

Initializing Local Variables

- Must generate explicit Code to Store the Values
- Among the procedure's first actions

C) Object-Oriented Languages

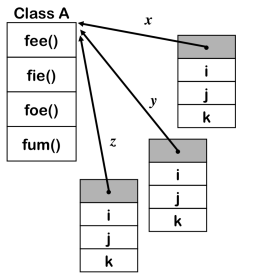
Class Structure

Note: See the slides for the method body structure

Method Code:

```
void A:fee(){
  ...
  self->i = 0;
  ...
}
```

```
t0 = &self + 4;
*t0 = 0;
```



objects of class A

Method Accesses Object Data as Offsets from the **self** Reference

Inheritance

See all the notes in the pdf of Run-Time Environments

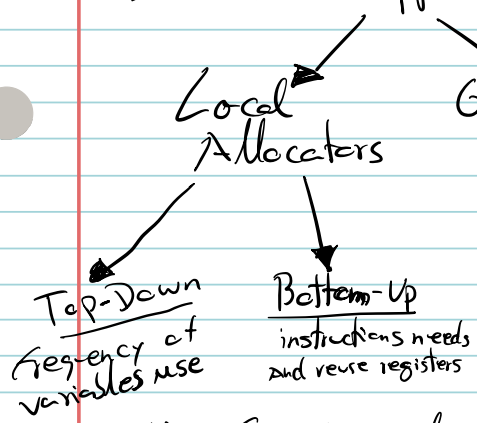
2. Register Allocation

- The significance and importance of register allocation (and assignment) a)
- Global and Local Allocation Algorithms (top-down and bottom-up) b)
- DU-chains and webs and their interference c)
- Global Register Allocation via graph coloring d)

a) RA is a part of the compiler's back end.
- Produces correct code that uses K registers.
- Minimizes space used to hold spilled values.
- is efficient.
- has a lot of impact

Importance
- Registers are one of the most critical Processor Resources.
- Reduces Memory Accesses that are super slow.
- Reduces N° of instructions, due to the register manipulation.

b) RA Approaches



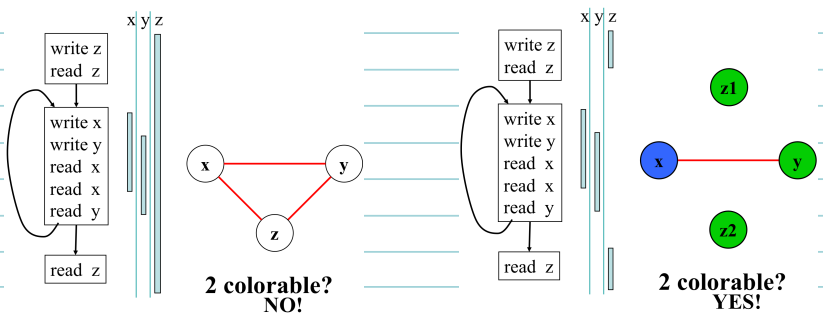
c) Webs combine DU-chains containing a common use.

Interference exists when live ranges overlap.

interference graph
nodes → Webs
edges → have interference

d) Graph Coloring

For each node, select a color so that their neighbors adjacent nodes have different colors comparing to him.



with Splitting

3. Control-Flow Analysis

present in 2. Register Allocation

- Definition and rationale of basic block construction.
- Dominators and an iterative algorithm to compute them b)
- Introduction to concepts of program optimization: constant propagation and the use of dominance for correctness reasoning. c)

b) Um nó x domina um nó y, se todos os caminhos possíveis entre o entry node e o nó y, incluiren o nó x.

Iterative Algorithm

- Make dominator set of the entry node itself.
- Make dominator set of the remainder nodes to be all graph nodes;
- (*) {
  - Visit the nodes in any order;
  - Make dominator set of the current node to be the intersection of the dominator sets of the predecessor nodes + the current node;
- Repeat until no change (*).

c) Optimizations:

- Constant Propagation;
- Algebraic Simplification;
- Copy Propagation;
- Common Sub-Expression Elimination (CSE);
- Dead Code Elimination;
- Loop Invariant Removal;
- Strength Reduction;
- Register Allocation.

Use of Dominance for Correctness Reasoning:

- Com os dominators conseguimos perceber o flow pretendido e ter controlo sobre ele porque para um bloco y executar, os seus dominators têm que executar primeiro.

4. Data-Flow Analysis

- What is Data-Flow Analysis and its general iterative algorithmic framework. a)
- Basic examples of available expressions and algorithmic implementation using the concepts of Gen and Kill sets as well as Transfer functions and control-flow merging functions. b)
- Basic properties of transfer functions: commutativity, associativity and the semi-lattice of values. Termination and speed. c)
- Live-Variable analysis and Copy-Propagation formulations as examples of Backward and Forward problem formulations. d)

a) The algorithm consists in IN and OUT Sets of the Basic Blocks that are formed from gen and Kill sets that form IN and OUT sets for each instruction.

Data flow Analysis is : *A collection of techniques for compile-time reasoning about the runtime flow of values in a program*

- Local Analysis
  - Analyze the "effect" of each Instruction in each Basic Block
  - Compose "effects" of instructions to derive information from beginning of basic block to each instruction
- Data-Flow Analysis
  - Iteratively propagate basic block information over the control-flow graph until no changes
  - Calculate the final value(s) at the beginning/end of the Basic Block
- Local Propagation
  - Propagate the information from the beginning/end of the Basic Block to each instruction

b) - Algorithm to compute Dominators.
- examples of algo's of available expressions in 4. Data-Flow Analysis and in the slides.
- Use-Def (UD) and Def-Use (DU) Chains. Example in the slides

c)
- Semi-Lattice/Lattice
  - Abstract quantities V over which the analysis will operate
  - Two operations meet ($\wedge$) and join ($\vee$) over values of V
  - A top value ($\top$) and a bottom value ($\bot$)
  - Example: Sets of Available Expressions and Intersection
- Transfer Functions
  - How each instruction (statement) and control-flow construct affects the abstract quantities V
    Example: the OUT equation for each statement
- Merging of Control-Flow Paths
  - Combining Operator of Data-Flow "meet" Operation
  - Typically Union or Intersection
  - *not the same as the lattice "meet" or "join"...*

In the slides

↓

A lot of theory

d) In the last section of 4. Data-Flow Analysis and Forward example in the slides