This exam asked the student to answer 4 out of 6 question. This document contains the 4 questions that were answered.

Question 1: N-Limited Constant Propagation

Consider a generalized kind of constant propagation called *N-limited* constant propagation. This analysis determines a *set* of up to N values for each variable *v* at each program point. The meaning is that *v* is guaranteed to have one of the N values at that point. (Normal constant propagation is 1-limited.)

Part (a):

Define a dataflow framework that defines N-limited constant propagation for a fixed N. Assume the usual simple imperative programming language (a program is a single procedure, there is no aliasing, etc).

Part (b):

How can the results of N-limited constant propagation be used by an optimizing compiler?

Part (c):

What are the advantages and disadvantages of this dataflow problem compared with normal constant propagation?

Question 2: Non-Local Gotos

Consider a language that allows *non-local gotos* as follows:

> A statement of the form "**goto** *L*" is legal iff, when the goto is executed, there is some procedure that has been called, has executed a statement labeled *L*, and has not yet returned (this procedure could be the one that contains the goto). If this is the case, control is transferred to the most recently executed such statement; otherwise, a runtime error occurs. Note that a procedure may have more than one statement with the same label.

Part (a):

Is it possible for the compiler to determine whether every goto in a program is legal? If yes, describe how it would be done. If no, explain why not, and give an analysis that would be safe but perhaps overly conservative (i.e., would never allow an illegal goto, but might say that a legal goto was illegal).

Part (b):

Describe how non-local gotos could be implemented at runtime (assume a compiled, not an interpreted language). Think about what special operations need to be implemented when a procedure is called, when a procedure returns (normally), when a labeled statement is executed, and when a goto is executed.

Question 3: Partial Evaluation and Denotational Semantics

Recall that the first step of partial evaluation is to compute a *division* by doing a binding-time analysis. For this question you will define a denotational semantics so that the meaning of a program is a uniform, congruent division (*i.e.*, a mapping from the identifiers that appear in the program to their bindings–either *static* or *dynamic*).

Here is the grammar for the language:

P → C.
C → C1 ; C2 | if B then C | if B then C1 else C2 | I := E | while B do C | read(I)
E → E1 + E2 | I | N
B → E1 = E2 | ! B | TRUE | FALSE
I → IDENT
N → NUM

The division defined by your denotational semantics should be the most optimistic division that satisfies the following properties (*i.e.*, as few variables as possible should be classified dynamic).

1. Every variable that occurs in a read statement is dynamic.
2. If the right-hand side of an assignment includes a dynamic variable, then the left-hand-side variable is dynamic.
3. If the condition of a while loop includes a dynamic variable, then every variable that is assigned to inside the loop is dynamic.

To specify the denotational semantics you will need to provide semantic algebras and valuation functions. The algebras and functions for the standard semantics, which defines the meaning of a program to be the final value of variable Z, are given on the next page to serve as a model for your answer. The given standard semantics does *not* include a rule for the read statement, but you should include that in your answer.

STANDARD DENOTATIONAL SEMANTICS

Semantic Algebras

1. Truth Values
   Domain
      Tr = {true, false}
   Operations
      not: Tr → Tr

2. Identifiers
   Domain
      i, i1 ε Id = Identifier

3. Natural Numbers
   Domain
      n ε Nat = {zero, one, two, ...}
   Operations
      plus: Nat → Nat → Nat
      equals: Nat → Nat → Tr

4. Stores
   Domain
      s, s' ε Store = Id → Nat
   Operations
      Newstore: Store = λ i. zero
      Access: Id → Store → Nat = λ i. λ s. s i
      Update: Id → Nat → Store → Store = λ i. λ n. λ s. λ i1. if i1 is i then n else s (i1)

Valuation Functions

P: Program → Nat → Nat
   P[C.] = λ n. let s = update A n Newstore in
                  let s' = **C**[C] s in Access Z s' endlet endlet

C: Command → Store → Store
   C[C1;C2] = λ s. **C**[C2] (**C**[C1] s)

C[if B then C] = λ s. if (**B**[B] s) then **C**[C] s else s
C[if B then C1 else C2] = λ s. if (**B**[B] s) then **C**[C1] s else **C**[C2] s
C[I := E] = λ s. Update I (**E**[E] s) s
C[while B do C] = fix(λ f. λ s. if **B**[B] s then f (**C**[C] s) else s)

E: Expression → Store → Nat
E[I] = λ s. Access I s
E[N] = λ s. N
E[E1 + E2] = λ s. (E[E1] s) plus (E[E2] s)

B: BooleanCondition → Store → Nat
B[TRUE] = λ s. true
B[FALSE] = λ s. false
B[!B] = λ s. not (**B**[B] s)
B[E1 = E2] = λ s. (**E**[E1] s) equals (**E**[E2] s)

Question 4: Partial Orders

This question concerns a pointed complete partial order (cpo) [Part (a)] or two pointed cpos [Part (b)].

Part (a):

Consider two chains $C = C_1 \sqsubseteq C_2 \sqsubseteq \ldots \sqsubseteq C_i \sqsubseteq \ldots$ and $D = D_1 \sqsubseteq D_2 \sqsubseteq \ldots \sqsubseteq D_j \sqsubseteq \ldots$ such that forall i there exists j such that $C_i \sqsubseteq D_j$, and forall j there exists i such that $D_j \sqsubseteq C_i$. Show that $\bigsqcup_{i=0}^{\infty} C_i = \bigsqcup_{j=0}^{\infty} D_j$.

Part (b):

Suppose that α and γ define a Galois insertion between Conc and Abs in the usual way; i.e.,

$$\alpha : \text{Conc} \rightarrow \text{Abs}$$
$$\gamma : \text{Abs} \rightarrow \text{Conc}$$

such that

$$a = \alpha(\gamma(a)), \text{forall } a \in \text{Abs}, \tag{1}$$

and

$$c \sqsubseteq \gamma(\alpha(c)), \text{forall } c \in \text{Conc}, \tag{2}$$

Assume that both γ and α are continuous.

Is it true that $\gamma(\bot_{Abs}) = \bot_{Conc}$? Either give a proof or a counter example.

Is it true that $\gamma(\top_{Abs}) = \top_{Conc}$? Either give a proof or a counter example.