Homework 3: Data Flow Analysis

Due: problems 1–9, February 18, 2008; problems 10-15, March 17, 2008.

In this homework you will learn more about data flow analysis. You will also learn more about how to do calculational style proofs.

Turn in on paper your answers, which have been typed on a computer (no handwritten answers, please).

If you wish, you can work in groups, although I don't recommend trying to do proofs in groups. Note that if you do work together, you must carefully follow the course grading policies.

Section 1.7: Algorithms

Read section 1.7 of our textbook: *Principles of Program Analysis* [4].

1. (20 points) [Concepts] [Calculate]

   Do exercise 1.6. In your proof use a calculational style (as in the handouts we provided in class [1, Section 4.2] [2, Chapter 4], see also Gries's article in *CACM* [3]), in which you justify each step.

2. (20 points) [Concepts] [Calculate]

   Do exercise 1.7. Again use the calculational style for any calculations or proofs involved.

   Here's an example, developed with Neeraj Khanolkar, which shows how to do such problems in calculational style. Consider the following equations among sets drawn from a universe U, where $a, b \in U$.

   $$\begin{array}{rcl} X_1 & = & \{a\} \\ X_2 & = & \{b\} \cup X_1 \cup X_2 \end{array}$$

   We can represent these equations as a function F as follows.

   $$\begin{array}{rcl} F & : & (U \times U) \to (U \times U) \\ F(u_1, u_2) & = & (F_1(u_1, u_2), F_2(u_1, u_2)) \\ F_1(u_1, u_2) & = & \{a\} \\ F_2(u_1, u_2) & = & \{b\} \cup u_1 \cup u_2 \end{array}$$

   We wish to solve these using Chaotic Iteration. Let us represent the steps of the Chaotic Iteration algorithm using the symbol $\leadsto$. Writing $\vec{u} = (x, y)$ to mean $u_1 = x \wedge u_2 = y$ (because when dealing with 12-tuples it will be convenient to have a smaller formula), we can calculate as follows.

   $$\begin{array}{ll} & \vec{u} = (\{\}, \{\}) \\ \leadsto & \quad \langle \text{by definition of } F_2, u_2 \neq F_2(\{\}, \{\}) \rangle \\ & \vec{u} = (\{\}, \{b\}) \\ \leadsto & \quad \langle \text{by definition of } F_1 \rangle \\ & \vec{u} = (\{a\}, \{b\}) \\ \leadsto & \quad \langle \text{by definition of } F_2 \rangle \\ & \vec{u} = (\{a\}, \{a, b\}) \end{array}$$

   At this point no more steps are possible, so the fixed point of F is $\vec{u} = (\{a\}, \{a, b\})$, i.e.,

   $$\begin{array}{rcl} X_1 & = & \{a\} \\ X_2 & = & \{a, b\} \end{array}$$

   is a solution to the equations above.

3. (25 points) [Concepts] [Calculate]

   Do exercise 1.8. Again use the calculational style for any calculations or proofs involved.

Section 2.1: Intraprocedural Analysis

Read section 2.1 of our textbook: *Principles of Program Analysis* [4].

4. [Concepts] [Calculate]

   This problem is about finding examples to illustrate what starting values is needed to compute a solution by iteration, for the data flow analyses in this section.

   (a) (10 points) According to the book, when using Chaotic iteration to solve for a solution of the Available Expressions analysis (in Section 2.1.1), one should start with a tuple in which each element is $\mathbf{AExp}_\star$. To see why this is necessary, create an example program in the WHILE language for which the Chaotic iteration of the function that represents that program's Available Expressions analysis does not get the right result, if the iteration starts with $\vec{\emptyset}$. (Hint: the next part of this problem will be easiest if your example is very small.)

   (b) (10 points) Show how, the Chaotic iteration for your example gives the right result for the Available Expressions analysis, if you start the iteration with a tuple in which each element is $\mathbf{AExp}_\star$.

   (c) (10 points) Similarly, give an example program in the WHILE language in which the chaotic iteration for the Very Busy Expressions analysis (of section 2.1.3) does not give the right result, if the iteration starts with $\vec{\emptyset}$.

   (d) (10 points) Show how, the Chaotic iteration for your example gives the right result for the Very Busy Expressions analysis, if you start the iteration with a tuple in which each element is $\mathbf{AExp}_\star$.

   (e) (10 points) What property of an analysis, in general, determines what starting value is appropriate for iterations used to find a solution?

   (f) (10 points; extra credit) Characterize the class of programs (perhaps syntactically) for which the starting value of an iteration makes no difference.

5. (suggested practice) Do exercise 2.1.

6. (15 points) [Concepts] [Calculate]

   Do exercise 2.2.

7. (20 points) [Concepts]

   Do exercise 2.3.

8. (25 points) [Concepts] [Calculate]

   Do exercise 2.4.

9. (120 points; extra credit) [Concepts] [Soundness]

   Do the first part of Mini Project 2.1. Use the calculational style for your proof. (Note that this is quite difficult to do in full detail!)

10. (50 points; extra credit) [Concepts]

    Do the third part of Mini project 2.1.

Section 2.2: Theoretical Properties

Read section 2.2 of our textbook: *Principles of Program Analysis* [4].

11. (20 points) [Concepts] [Soundness]

    Do a proof of part (iv) of lemma 2.14 using the calculational style in your proof.

12. [Concepts] [Semantics]

    Write additions to table 2.6 to handle the following new pieces of syntax. (Hint, all of these can be handled without changing the configurations, and in particular you do not need to introduce "errors" or $\perp$ to the domain **State**.)

    (a) (5 points) Nondeterministic (demonic) choice statements of the form $S_1 \,[]\, S_2$, whose meaning is to execute either S_1 or S_2. (Hint: use 2 rules.)

    (b) (5 points) Parallel execution statements of the form $S_1 \mid\mid S_2$, whose meaning is to execute both S_1 and S_2 in an interleaved fashion.

    (c) (10 points) Assume statements of the form `assume` b, whose meaning is to do nothing if b is true, but to refuse to execute if b is not true. That is, the configuration can be "stuck" when b is not true.

    (d) (5 points) A `break` statement. This breaks out of the closest surrounding `while` loop. You can assume that `break` statements only occur inside `while` loops. Hint: you can use "context" in operational semantics rules by only allowing a `break` to execute in certain positions of a configuration.

    (e) (5 points) A `return` statement, which takes no arguments (as in `void` methods in Java) and simply halts execution without changing the state. Hint: you may want to think about changing the sequence rules.

    (f) (10 points) A `throw` statement, and a `try-catch` statement of the form `try` S_1 `catch` S_2, where S_1 and S_2 are statements. where you can assume that all `throw` statements occur inside a `try-catch` statement, and in which a `throw` statement jumps (without changing the state) to the `catch` block of closest surrounding `try-catch` statement.

13. (30 points; extra credit) [Concepts] [Semantics] [Soundness]

    Which of the additions to the WHILE language in the previous problem, if any, invalidate Lemma 2.14? Give a counterexample for any parts invalidated, and use the calculational style to show how the counterexample works out. Conversely, prove the parts not invalidated by various additions using the calculational style.

14. (20 points) [Concepts]

    Do the second part of Mini Project 2.1. In your proof of correctness, assume that the formulation of the UD analysis given in section 2.1.5 (just before section 2.2) of the textbook is correct. This will permit an easy proof of correctness for a constructive definition of the DU analysis that builds on the (assumed) correctness of the UD analysis.

    Use the calculational style for your proofs.

15. (80 points) [Semantics] [Soundness]

    Do Mini Project 2.2. Use the calculational style for your proofs.

References

[1] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer-Verlag, 1998.

[2] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and program semantics*. Springer-Verlag, NY, 1990.

[3] David Gries. Teaching calculation and discrimination: A more effective curriculum. *Communications of the ACM*, 34(3):44–55, March 1991.

[4] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag, second printing edition, 2005.