Introduction.
Textual Substitution, Equality, and Assignment
CS 2LC3

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Basic Information

Instructor: Dr. Ryszard Janicki, ITB 217, e-mail: janicki@mcmaster.ca, tel: 525-9140 ext: 23919
Teaching Assistants:

- Mahdee Jodayree: mahdijaf@yahoo.com
- Habib Ben Abdallah: benabdh@mcmaster.ca
- Mingzhe Wang: wangm235@mcmaster.ca
- Zaid Alnemer: alnemez@mcmaster.ca
- Luna Yao: yaol13@mcmaster.ca

Course website: `http://www.cas.mcmaster.ca/~cs2lc3`,
Lectures: Monday: 12:30pm - 1:20pm, Tuesday: 1:30pm - 2:20pm, Thursday 12:30pm - 1:20pm, All in JHE 376
Tutorials: Friday: 8:30am -10:20am, ITB 139, Wedn.: 12:30pm-2:20pm, T13 105, Tuesday 2:30pm-4:30pm, JHE A101, Friday 12:30pm - 2:20pm, ITB 139 Wednesday: 11:30am-1:20pm, BSB 120, start Sept 13, 2022
Office hours: Monday 1:30-2:20 pm
Midterm: Thursday, October 20, in JHE 376, during class time (tentatively)

Course Details

Calendar Description:
Introduction to logic and proof techniques for practical reasoning: propositional logic, predicate logic, structural induction; rigorous proofs in discrete mathematics and programming.

Goals:
This course will teach logical formalisation and reasoning skills as tools intended ultimately for system specification and for correctness arguments. To a large degree, this can be seen as analogous to acquiring language skills, including knowledge and skills concerning syntax, semantics, pragmatics, and vocabulary of the language of logical reasoning and of discrete mathematics, which can be seen as the mathematics of data structures and of software correctness. Conscious and precise use of this language is the foundation for precise specification and rigorous reasoning, which take a central place in this course.

Learning Objectives

Preconditions:

Students should have basic knowledge of discrete mathematics know how to program in at least one programming language.

Postconditions:

1. Students should confidently write syntactically and type-correct logical formulae.
2. Students should confidently perform calculational proofs in propositional logic.
3. Students should confidently produce structured and calculational proofs in predicate logic applications.
4. Students should understand and routinely use induction proofs, including structural induction.
5. Students should confidently formalize natural-language specifications.
6. Students should confidently reason with and about discrete structures such as sets, functions, relations, graphs.
7. Students should be able to prove correctness properties of imperative programs.

Outline and Texts

Course Outline (Tentative, there might be slight changes):

1. Introduction to Calculational Reasoning; Parts of Chapters 1, 15
2. Boolean Expressions and Propositional, Logic; Chapters 1-5
3. Quantification, Predicate Logic, Sets; Chapters 8-9, 11
4. Induction, Sequences, Trees; Chapters 12-13
5. Relations and Functions, Graphs Chapters; 14, 19
6. Correctness of Imperative Programs; Chapter 10, other sources

Text: David Gries and Fred B. Schneider: A Logical Approach to Discrete Math, Springer 1993, ISBN 3-540-94115-0

The course will not always follow the text-book closely, and some extra material will be given on the course website.

Evaluation

- **Evaluation:** The will be a 2.5 hours (take home) final examination (50%), 60 minutes midterm test (17%, take home during the class time) and three assignments ($3 \times 11 = 33\%$).
- **Detailed grading scheme:** $Grade = 0.50 \times exam + 0.17 \times midterm + 0.11 \times (assg1 + assg2 + assg3)$
- *Late assignments will not be accepted as solutions will be posted almost immediately after deadlines.*
- Although you may discuss the general concept of the course material with your classmates, your assignment must be your individual effort.

Conventional Proof of $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

We first show that $A \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C)$. If $x \in A \cup (B \cap C)$, then either $x \in A$ or $x \in B \cap C$. If $x \in A$, then certainly $x \in A \cup B$ and $x \in A \cup C$, so $x \in (A \cup B) \cap (A \cup C)$. On the other hand, if $x \in B \cap C$, then $x \in B$ and $x \in C$, so $x \in A \cup B$ and $x \in A \cup C$, so $x \in (A \cup B) \cap (A \cup C)$. Hence, $A \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C)$.

Conversely, if $y \in (A \cup B) \cap (A \cup C)$, then $y \in A \cup B$ and $y \in A \cup C$. We consider two cases: $y \in A$ and $y \notin A$. If $y \in A$, then $y \in A \cup (B \cap C)$, and this part is done. If $y \notin A$, then, since $y \in A \cup B$ we must have $y \in B$. Similarly, since $y \in A \cup C$ and $y \notin A$, we have $y \in C$. Thus, $y \in B \cap C$, and this implies $y \in A \cup (B \cap C)$. Hence$(A \cup B) \cap (A \cup C) \subseteq A \cup (B \cap C)$.

Calculational Proof of $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

Proof.

$x \in A \cup (B \cap C)$

$\iff$ $\langle$ Definition of $\cup$ $\rangle$

$x \in A \vee x \in B \cap C$

$\iff$ $\langle$ Definition of $\cap$ $\rangle$

$x \in A \vee x \in B \ \wedge \ x \in C$

$\iff$ $\langle$ Distributivity of $\vee$ over $\wedge$ $\rangle$

$(x \in A \vee x \in B) \ \wedge \ (x \in A \vee x \in C)$

$\iff$ $\langle$ Definition of $\cup$, twice $\rangle$

$x \in (A \cup B) \ \wedge \ x \in (A \cup C)$

$\iff$ $\langle$ Definition of $\cap$ $\rangle$

$x \in (A \cup B) \cap (A \cup C)$

■

Conventions and notation

- The last presentation of the proof is obvious and straightforward
- Anyone with a little experience in such calculational proofs will have no difficulty reproducing them
- These proofs are rigorous and could be checked by a mechanical proof checker

Conventions and notation

$$
\begin{array}{ll}
 & expression_0 \\
op_0 & \langle\ hint_0\ \rangle \\
 & expression_1 \\
op_1 & \langle\ hint_1\ \rangle \\
 & expression_2 \\
\cdots & \langle\ \cdots\ \rangle \\
 & expression_n
\end{array}
$$

where op_i, $i = 0, \cdots, n-1$, is a relational operator

$$=, <, >, \leq, \geq, \sqsubseteq, \sqsupseteq, \cdots$$

like it could be a logical operator

$$\Longrightarrow \quad \text{and} \quad \Longleftrightarrow \ .$$

Example

Prove that $2 * (5 * x^2 - 2 * x + 6 * x + X^2 - 4) = 12 * X^2 + 8(X - 1)$

$$
\begin{array}{ll}
 & 2 * (5 * x^2 - 2 * x + 6 * x + X^2 - 4) \\
= & \quad \langle \text{ Distributivity of * over } + \quad \& \quad \text{calculus } \rangle \\
 & 10 * x^2 - 4 * x + 12 * x + 2 * X^2 - 8 \\
= & \quad \langle \text{ Commutativity of } + \quad \& \quad \text{calculus } \rangle \\
 & 12 * X^2 + 8 * x - 8 \\
= & \quad \langle \text{ Distributivity of * over } + \rangle \\
 & 12 * X^2 + 8 * (x - 1)
\end{array}
$$

Example

$$
\begin{array}{ll}
 & 3x + 3 = 0 \\
\Longleftrightarrow & \quad \langle \text{ Add } -3 \text{ in the two sides of the equation} \\
 & \quad \& \quad 3 - 3 = 0 \quad \& \quad 0 \text{ is the neutral element for } + \rangle \\
 & 3x = -3 \\
\Longleftrightarrow & \quad \langle \ 3 \neq 0 \quad \& \quad \text{divide by 3 the two sides of the equation} \quad \& \quad \tfrac{3}{3} = 1 \ \rangle \\
 & x = -1
\end{array}
$$

When appropriate, we adopt this calculational way to present proofs.

Expressions : Syntax

Syntax of Conventional Mathematical Expressions:

- A constant (e.g. 231) or variable (e.g. x) is an expression.
- If E is an expression, then (E) is an expression.
- If $\circ$ is a unary prefix operator and E is an expression, then $\circ E$ is an expression, with operand E.
  For example, the negation symbol $-$ is used as a unary operator, so -5 is an expression.
- If $*$ is a binary infix operator and D and E are expressions, then $D * E$ is an expression, with operands D and E.
  For example, the symbols $+$ (for addition) and $\times$ (for multiplication or product) are binary operators, so $1 + 2$ and $(-5) \times (3 + x)$ are expressions.

Expressions : Evaluation

- A *state* is simply a list of variables with associated values. For example, in the state consisting of $(x, 5)$ and $(y, 6)$, variable x is associated with the value 5 and variable y with 6.
- *Evaluation of an expression* E in a state is performed by replacing all variables in E by their values in the state and then computing the value of the resulting expression.
  For example, evaluating $x - y + 2$ in the state $\{(x, 5), (y, 6)\}$ consists of replacing variables x and y by their values to yield $5 - 6 + 2$ and then evaluating that to yield 1 .

Textual Substitution

- Let E and R be expressions and let x be a variable. We use the notation $E[x := R]$ or E^x_R to denote an expression that is the same as E but with all occurrences of x replaced by "(R)" .
- The act of replacing all occurrences of x by "(R)" in E is called textual substitution.
- Examples:

Expression	Result	Unnecessary parentheses removed
$x[x := z + 2]$	$(z + 2)$	$z + 2$
$(x + y)[x := z + 2]$	$((z + 2) + y)$	$z + 2 + y$
$(x \cdot y)[x := z + 2]$	$((z + 2) \cdot y)$	$(z + 2) \cdot y$

Inference Rule Substitution

- Inference Rule (in general):

$$\frac{P_1, \ldots, P_k}{C},$$

  where: P_i - *premises* or *hypotheses*, C - conclusion
- Inference rule asserts that if the premises are theorems, then the conclusion is a theorem.
- Inference rule *Substitution*: E-expression, v - list of variables, F - list of expressions

$$\frac{E}{E[v := F]}.$$

- Example:

$$\frac{2 \cdot x/2 = x}{(2 \cdot x/2 = x)[x := j + 5]} \text{ or } \frac{2 \cdot x/2 = x}{2 \cdot (j + 5)/2 = j + 5}.$$

Textual Substitution and Equality

- Evaluation of the expression $X = Y$ in a state yields the value *true* if expressions X and Y have the *same value* and yields *false* if they have *different values*.
- Laws:
  - **Reflexivity:** $x = x$
  - **Symmetry:** $(x = y) = (y = z)$
  - **Transitivity:** $\dfrac{X = Y, Y = Z}{X = Z}$
  - **Leibnitz:** $\dfrac{X = Y}{E[z := X] = E[z := Y]}$
- Example of Leibnitz: Assume $b + 3 = c + 5$ is a theorem. Then $d + b + 3 = d + c + 5$ is a theorem by Leibnitz with $X : b + 3$, $Y : c + 5$, $E : d + z$ and $z : z$.

Assignment Statement

- Execution of the **assignment statement**

  $$x := E$$

  evaluates expression E and stores the result in variable x.
  Assignment $x := E$ is read as "x *becomes* E".
- Execution of $x := E$ in a *state* stores in x the value of E in that state, thus changing the state.
- *Example.* Suppose the state consists of $(v, 5), (w, 4), (x, 8)$ and consider the assignment $v := v + w$.
  The value of $v + w$ in the state is 9 , so executing $v := v + w$ stores 9 in v , changing the state to $(v, 9), (w, 4), (x, 8)$.

Preconditions and Postconditions

- A **precondition** of a statement is an assertion about the program variables in a state in which the statement may be executed.
- A **postcondition** is an assertion about the states in which it may terminate.
- *Hoare Triple* - a notation:

$$\{P\}\ S\ \{Q\}$$

where S - *statement*, P - *precondition*, Q - *postcondition*.

- Examples of valid Hoare triples:

$$\begin{aligned} \{x = 0\} \quad x := x + 1 \quad \{x > 0\} \\ \{x > 5\} \quad x := x + 1 \quad \{x > 0\} \\ \{x + 1 = 0\} \quad x := x + 1 \quad \{x > 0\} \end{aligned}$$

- Invalid Hoare triple:

$$\{x = 5\} \quad x := x + 1 \quad \{x = 7\}$$

Definition of Assignment as Hoare Triple

- **Assignment as Hoare Triple**

$$\{R[x := E]\}\ x := E\ \{R\}$$

Example: Assume: E is $x := x + 1$, R is $x > 4$. Then $\{R[x := E]\}$ is $(x > 4)[x := x + 1]$ which is $x > 4$. Hence :

$$(x > 4)\ x := x + 1\ (x > 5).$$

- More examples:

$$\begin{array}{rll} \{x + 1 > 5\} & x := x + 1 & (x > 5) \\ \{5 \neq 5\} & x := 5 & \{5 \neq 5\} \\ \{x^2 > x^2 \cdot y\} & x := x^2 & \{x > x \cdot y\} \end{array}$$

Calculation of Hoare Triples

- The tendency is to expect the postcondition to be calculated from the precondition and to expect the definition to be $\{R\}\ x := E\ \{R[x := E]\}$.
- Fight this intuition, for it is not consistent with how the assignment is executed. For example, using this incorrect rule, we would obtain $\{x = 0\}\ x := 2\ \{(x = 0)[x := 2]\}$, which is invalid.
- This is because when the assignment terminates, the resulting state ($x = 2$ in this case) does not satisfy the postcondition ($x = 0$ in this case), false.

Multiple Assignments

- Multiple assignment (used mainly in theory)

$$x_1, x_2, \ldots, x_n := E_1, E_2, \ldots, E_n,$$

  where the x_i are distinct variables and the E_i are expressions.
- Execution of multiple assignments: First evaluate all the expressions E_i to yield values v_i; then assign v_1 to x_1, v_2 to x_2, ... , and finally v_n to x_n.
- Note that all expressions are evaluated before any assignments are performed.
- Examples:

$x, y := y, x$	Swap x and y
$x, i := 0, 0$	Store 0 in x and i
$i, x := i + 1, x + i$	Add 1 to i and i to x
$x, i := x + i, i + 1$	Add 1 to i and i to x

- Note that $x_1, x_2 := E_1, E_2$ may differ from $x_1 := E_1; x_2 := E_2$, for example $x, y := x + y, x + y$ differs from $x := x + y; y := x + y$.