CPSC 121: Models of Computation

Unit 5
Predicate Logic

Based on slides by Patrice Belleville and Steve Wolfman

Before-Class Learning Goals

- By the start of class, you should be able to
  - Evaluate the truth of predicates applied to particular values.
  - Show a predicate logic statement is true by enumerating examples (i.e., all in the domain/one for a universal/existential quantifier).
  - Show a predicate logic statement is false by enumerating counterexamples (i.e., one/all in the domain for a universal/existential quantifier).
  - Translate between statements in formal predicate logic notation and equivalent statements in closely matching informal language (i.e., informal statements with clear and explicitly stated quantifiers).

Quiz 5 Feedback

- Overall:
- Specific Issues:

- We will discuss the open-ended question on what it means for an algorithm to be faster/slower than another one next week.

In-Class Learning Goals

- By the end of this unit, you should be able to:
  - Build statements about the relationships between properties of various objects using predicate logic.
  - These may be
    - real-world like “every candidate got votes from at least two people in every province” or
    - computing related like “on the i-th repetition of this algorithm, the variable min contains the smallest element in the list between element 0 and element i”.

?Related to CPSC 121 Big Questions ?

- How can we convince ourselves that an algorithm does what it's supposed to do?
  - We need to prove that it works.
  - Now we will learn how to model such problems in predicate logic using predicates and variables.
- How do we determine whether or not one algorithm is better than another one?
  - We can finally model and answer that question!

Unit Outline

- Predicates vs Propositions
- Examples
- More examples: sorted lists
- Algorithm efficiency revisited.
- Additional examples to consider.

Is Propositional Logic a complete model?

- Which of the following can propositional logic model effectively?
  - A. Relationships among factory production lines (e.g., "wheel assembly" and "frame welding" both feed the undercarriage line).
  - B. Defining what it means for a number to be prime.
  - C. Generalizing from examples to abstract patterns like "everyone takes off their shoes at airport security".
  - D. It can model all of these effectively.
  - E. It can not model any of these effectively.

What Predicate Logic is Good for

- Predicate logic is good for modeling:
  - Relationships among real-world objects
  - Generalizations about patterns
  - Infinite domains
  - Generally, problems where the properties of the different concepts, or parts, depend on each other
  - and more….

Predicate Logic in Computer Science

- Examples of predicate logic use in CS:
  - Data structures: **Every** key stored in the left subtree of a node N is smaller than the key stored at N (CPSC 221).
  - Language definition: No path via references exists from **any** variable in scope to **an**y memory location available for garbage collection... (CPSC 312)
  - Databases: the relational model is based on predicate logic (CPSC 304).
  - Algorithms: in the worst case, **every** comparison sort requires at least $c*n*\log_2 n$ comparisons to sort n values, **for some** constant $c > 0$ (CPSC 320).

Quantifier Scope

- A quantifier applies to everything to its right, up to the closing parenthesis of the () pair that "contains" it.
- Example:

$\forall x \in D, (\exists y \in E, Q(x, y) \rightarrow \forall z \in F, R(x, z)) \wedge P(x)$

Quantifier Scope (cont')

- Which of the following placements of parentheses yields the same meaning as:

$\forall x \in Z, \exists y \in Z, x < y \wedge Even(y)$?

A. $(\forall)x \in Z, \exists y \in Z, x < y \wedge Even(y)$

B. $(\forall x) \in Z, \exists y \in Z, x < y \wedge Even(y)$

C. $(\forall x \in Z), \exists y \in Z, x < y \wedge Even(y)$

D. $(\forall x \in Z, \exists y \in Z, x < y) \wedge Even(y)$

E. $(\forall x \in Z, \exists y \in Z, x < y \wedge Even(y))$

Negation Scope:

- Which of the following placements of parentheses yields the same meaning as:

$\sim\exists x \in Z+, \forall y \in Z+, x < y \wedge Even(y)$?

A. $(\sim\exists)x \in Z+, \forall y \in Z+, x < y \wedge Even(y)$

B. $(\sim(\exists x)) \in Z+, \forall y \in Z+, x < y \wedge Even(y)$

C. $(\sim(\exists x \in Z+)), \forall y \in Z+, x < y \wedge Even(y)$

D. $(\sim(\exists x \in Z+, \forall y \in Z+, x < y)) \wedge Even(y)$

E. $(\sim(\exists x \in Z+, \forall y \in Z+, x < y \wedge Even(y)))$

Predicates vs. Propositions

- What is the difference between a proposition and a predicate?
  - A. A predicate may contain one or more quantifiers, but a proposition never does.
  - B. A proposition's name is a lowercase letter, whereas a predicate's name is an uppercase letter.
  - C. A predicate may contain unbound variables, but a proposition does not.
  - D. They are the same thing, using different names.
  - E. None of the above.

Unbound Variables

- What is the truth value of the following formula?

  $\exists x \in Z,\ x*x = y$.

  - A. True, because (for example) 5*5=25.
  - B. True, because every `y=(sqrt y)*(sqrt y)`
  - C. False, because of counterexamples like no integer multiplied by itself equals 3.
  - D. It depends on y, but given a value for y, we could calculate a truth value.
  - E. None of the above.

Unbound Variables

- Which variables does this formula's truth depend on?

  $\forall i \in Z^+, (i \geq n) \rightarrow \sim\exists v \in Z^+, \text{HasValue}(a, i, v)$

  - A. i and v
  - B. a and n
  - C. n and v
  - D. i and n
  - E. None of these are correct.

Defining Predicates

- A ***predicate*** is a predicate logic formula with unbound variables

  `PerfectSquare(y): ∃x ∈ Z, x*x = y`

  where `y ∈ Z`
- Then
  - `PerfectSquare(25) is _____`
  - `PerfectSquare(3) is _____`
  - `∀y ∈ Z, PerfectSquare(y) is _____`
  - `∃y ∈ Z, PerfectSquare(y) is _____`

Unit Outline

- Predicates vs Propositions
- Examples
- More examples: sorted lists
- Algorithm efficiency revisited.
- Additional examples to consider.

Example 1

(c) Walt Disney Co.

- Given the definitions:
  - F: the set of foods.
  - E(x): Alice eats food x.
  - g: Alice grows.
  - s: Alice shrinks.
- Express these statements using predicate logic:
  - Eating food causes Alice to grow or shrink.
  - Alice shrank when she ate some food.

Example 2

(c) animal.discovery.com

- Given the definitions:
  - **D:** the set of all creatures.
  - **F(x)**: x is a fierce creature.
  - **L(x)**: x is a lion
  - **C(x)**: x drinks coffee
  - **T(x,y)**: creature x has "tasted" creature y.
- Express these statements using predicate logic:
  - All lions are fierce.
  - Some lions do not drink coffee.

Restricting the Domain of an Existential

Compare and contrast the following:

- Some creature drinks coffee:
- Some lion drinks coffee:
- Some fierce lion drinks coffee:

Restricting the Domain of a Universal

Compare and contrast the following:

- All creatures drink coffee:
- All lions drink coffee:
- All fierce lions drink coffee:

Ambiguity with Negation

- Consider the statements
  1. All fierce creatures are not lions
  2. Not all fierce creatures are lions
  - Their translations into predicate logic are:

  1.

  2.

  - Do we often mean (2) when we say (1)?

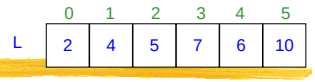
Order of Quantifiers

- Express these two propositions in English:
  - $\forall x \in D, \exists y \in D, T(x,y)$
  - $\exists x \in D, \forall y \in D, T(x,y)$
- Give an example where one of the propositions is true, and the other proposition is false.

Unit Outline

- Predicates vs Propositions
- Examples
- More examples: sorted lists
- Algorithm efficiency revisited.
- Additional examples to consider.

Example: Lists



- Definitions:
  - Assume that L represents a list of values.
  - The length of L is denoted by (length L).
  - The i-th element of L is denoted by (list-ref L i).
    - The first element of L is (list-ref L 0).
- Are length and list-ref predicates?
  - No: a predicate is a function that returns true or false.
  - What do these functions return?
    - length: an integer.
    - list-ref: a value whose type depends on the contents of L.

Lists (cont')

- Problem:
  - Define a predicate Sorted(L) whose value is true if and only if L is sorted in non-decreasing order.
  - We can use the functions length and list-ref.
- Assumption:
  - The call (list-ref L i) returns an undefined value if i is negative, or greater than or equal to (length L).
    - Recall

      the first element of L is (list-ref L 0),

      L's last element is (list-ref L (- (length L) 1)).

Lists (cont')

- Which of the following is/are a problem with this definition?

  Sorted(L) ≡ ∀i ∈ N, ∀j ∈ N, (list-ref L i) ^ (list-ref L j) ^ v1 < v2

  A. There is no quantifier for L.

  B. There are no quantifiers for v1 and v2.

  C. We can not use ^ with (list-ref L i) and (list-ref L j)

  D. Both (a) and (b)

  E. Both (b) and (c)

Lists (cont')

- Which of the following is a problem with this definition?

  Sorted(L) ≡ ∀i ∈ N, ∀j ∈ N, i < j →(list-ref L i) < (list-ref L j)

  A. It is too restrictive (it does not allow for equal values).

  B. It does not restrict the ranges of i and j.

  C. It is missing quantifiers.

  D. Both (a) and (b)

  E. Both (b) and (c)

Lists (cont')

- How do we modify the attempt on the previous slide to get a working predicate?

Sorted(L) ≡ ∀i ∈ N ∀j ∈ N, i < j

→ (list-ref L i) < (list-ref L j)

"At Most One", “Exactly One” and "At least Two"

- There exists means there is at least one.
- How do we write there is exactly one?
  - lists have exactly one element at each valid index.
- Definitions:
  - There is exactly one ≡
    There is at least one ∧ There is at most one.
  - There is at most one with property P ≡
    ∀x ∈ D, ∀y ∈ D, P(x) ∧ P(y) → x = y.
  - There is exactly one with property P ≡
    ∃x ∈ D, P(x) ∧ (∀y ∈ D, P(y) → x = y).
  - There are at least two ≡
    ∃x ∈ D, ∃y ∈ D, x ≠ y ∧ P(x) ∧ P(y).

Idiom Summary

“None…”/“No x…”	~∃x ∈ D, …
“At least one…”/“Some…”/ “A (particular) x…”	∃x ∈ D, …
“Every…”/“All…”/“Any x…”/ “A (arbitrary) x…”	∀x ∈ D, …
“Some P-ish x…” (restricting the domain)	∃x ∈ D, P(x) ∧ …
“Every P-ish x…” (restricting the domain)	∀x ∈ D, P(x) → …
“At least two…”	∃x ∈ D, ∃y∈D, x≠y ∧ …
“At most one with property P”	∀x ∈ D, ∀y ∈ D, P(x) ∧ P(y) → x = y.
“Exactly one with property P”	∃x ∈ D, P(x) ∧ (∀y ∈ D, P(y) → x = y)

Formal vs. Informal Reasoning

- Soon we will use English more often than writing every predicate explicitly using logic.
- However the ability to use predicate logic will help us think things through and not overlook minor (but important) details.
  - “when we become comfortable with formal manipulations, we can use them to check our intuition, and then we can use our intuition to check our formal manipulations.” -- Epp, (3rd ed), p. 106-107

Unit Outline

- Predicates vs Propositions
- Examples
- More examples: sorted lists
- Algorithm efficiency revisited.
- Additional examples to consider.

Algorithm Efficiency

- What does it mean for one algorithm to be generally faster than another algorithm?
- Here are some of the answers we have seen on the quiz:

Example

- Consider the following problem:
  - Given a sorted list of names with telephone numbers.
  - We want to find the phone number for a given name N.
- Which algorithm is generally faster?
  - A. Algorithm L: check the first name. If it's not N, then check the second name. Then the third name, etc.
  - B. Algorithm B: check the name in the middle of the list. If N comes earlier alphabetically, then search the first half of the list using B. If it comes later, search the second half of the list instead. Repeat until you have found N.

Example (cont')

- Assumptions:
  - Reading the name after the current name takes 1s on average.
  - Reading a name given its position takes 10s on average.
- For a list with 15 names:
  - Algorithm L takes 15 * 1s = 15s in the worst case.
  - Algorithm B takes 5 * 10s = 50s in the worst case.

Example (cont')

- For a list with 63 names:
  - Algorithm L takes 63 * 1s = 1m 3s in the worst case.
  - Algorithm B takes 7 * 10s = 1m 10s in the worst case.
- For a list with 1048575 names:
  - Algorithm L takes 1048575 * 1s = 12d 3h 16m 15s in the worst case.
  - Algorithm B takes 21 * 10s = 3m 30s in the worst case.

Comparing Algorithms

- How do we determine whether or not an algorithm is generally faster than another?
  - We want to measure how good the algorithm is, in a way that does not depend on
    - the programming language used to implement it.
    - the quality of the compiler or interpreter.
    - the speed of the computer it is executed on.
  - One idea is to count the number of **elementary steps** of the algorithm as a function of the size of its input n.
    - An elementary step is anything that can be computed in constant time, that is, independent from n.

Comparing Algorithms

- Is an algorithm with 3n steps faster than one with 6n steps?
  - A. Yes, always.
  - B. No, never.
  - C. Sometimes.
  - D. None of the above.

Comparing Algorithms

- Example:
  - One algorithm performs 6n steps of the following type (only the first 6 are written):

3 + 8	2 + 4	6 + 9
2 + 11	5 + 6	7 + 1

  - The other algorithm performs 3n steps of the following type (only the first 3 are written):

$$\int_2^5 \frac{x^4 - x^2}{2}\,dx \qquad \int_1^6 x\cos(x) + \sin(x)\,dx \qquad \int_1^\infty \frac{1}{x^2}\,dx$$

  - Which one is faster?

Comparing Algorithms

- Facts about execution times:
  - we can not rely on the values of the constants in front of the functions describing the number of steps.
  - it's almost impossible to compute the number of steps exactly.
- So we want to come up with
  - a way to count step that ignores these constants.
  - an approximation of the correct number of steps.

Defining Algorithm Efficiency

- Terminology: an algorithm runs in O(g) time, stated "big-Oh of g time", if it executes (approximately) at most g(n) steps.
- Examples:
  - Algorithm L runs in O(n) time.
  - Algorithm B runs in $O(\log_2 n)$ time.
  - The algorithm we used to order students by date of birth runs in $O(n^2)$ time.
- Let's see how we can define O more precisely using quantifiers.

Defining Algorithm Efficiency

- Which of the following predicates says that the number of steps f(n) executes is (approximately) at most n^2?

A. $\forall c \in R^+ \; \forall n \in N \;\; f(n) \leq c\, n^2$

B. $\exists c \in R^+ \; \exists n \in N \;\; f(n) \leq c\, n^2$

C. $\exists c \in R^+ \; \forall n \in N \;\; f(n) \leq c\, n^2$

D. $\forall c \in R^+ \; \exists n \in N \;\; f(n) \leq c\, n^2$

E. None of the above.

Defining Algorithm Efficiency

- For which of the following functions f(n) is the predicate from the previous slide true?

A. $f(n) = n$

B. $f(n) = n^2/2$

C. $f(n) = 3n^2$

D. $f(n) = 2^n$

E. All of them

Defining Algorithm Efficiency

- Which of the following two functions grows faster?
  - A. f(n) = n
  - B. f(n) = $n\log_2 n$
  - C. Neither; they both grow equally fast.
- Is the following predicate true for f(n) = n?

  $\exists c \in R^+ \; \forall n \in N \;\; f(n) \leq cn\log_2 n$

  - A. Yes
  - B. No

Defining Algorithm Efficiency

- Is the following predicate true for f(n) = n?

  $\exists c \in R^+ \; \exists n_0 \in N \;\; \forall n \in N \;\; n \geq n_0 \rightarrow f(n) \leq cn\log_2 n$

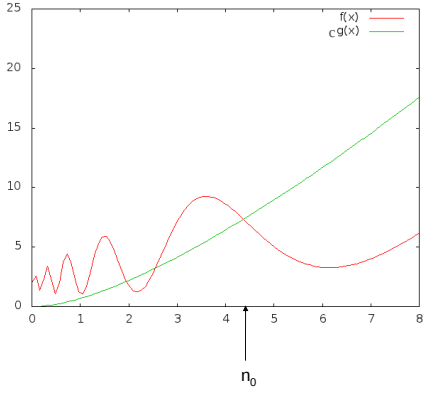
  - A. Yes
  - B. No
- So we define **O(g)** by:

  f is in O(g) if

  $\exists c \in R^+, \exists n_0 \in N, \;\; \forall n \in N, \;\; n \geq n_0 \rightarrow f(n) \leq cg(n)$

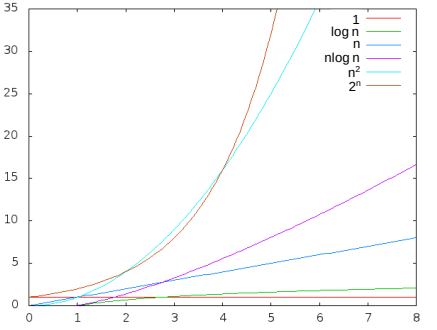
Defining Algorithm Efficiency

- Pictorially:



Common Running Times

- Some common running times:



Revisiting Sorted Lists

- Recall
  Sorted(L) ≡
  ∀i ∈ N, ∀j ∈ N, (0 ≤ i) ^ (i < j) ^ (j < (length L)) →
  (list-ref L i) ≤ (list-ref L j)
- If we verify that L is sorted using this definition, how many comparisons will we need?
- Can we do better?

Revisiting Sorted Lists

- Here is another definition:
  Sorted(L) ≡
  ∀i ∈ N, (0 ≤ i) ^ (i < (length L) - 1) →
  (list-ref L i) ≤ (list-ref L i+1)
- These two definitions are logically equivalent.
- If we verify that L is sorted using this definition, how many comparisons will we need?

Unit Outline

- Predicates vs Propositions
- Examples
- More examples: sorted lists
- Algorithm efficiency revisited
- Additional examples to consider

Unit 5: Predicate Logic

- Specifying the behaviour of a function/method that takes a list L and a value x:
  - Translate "returns true if and only if either L and x are both equal to null, or L contains at least one element e that is equal to x".
- Define a predicate Prime(x) that evaluates to true if and only if x is a prime. Assume that you have a predicate | such that x | y is true if and only if x divides y (that is, y/x is an integer).

Reading for Quiz #6

- Online quiz #6 is tentatively due _____.
- Reading for the quiz:
  - Epp, 4th edition: 3.2, 3.4
  - Epp, 3rd edition: 2.2, 2.4
  - Rosen, 6th edition: 1.3, 1.4
  - Rosen, 7th edition: 1.4, 1.5