Propositional and Predicate Logic

CS 536: Science of Programming, Fall 2022

ver Tue 2023-01-10, 14:35

A. Why

- Reviewing/overviewing logic is necessary because we'll be using it in the course.
- We'll be using predicates to write specifications for programs.
- Predicates and programs have meaning relative to states.

B. Outcomes

At the end of this class, you should

- Be able to prove simple logical equivalences of propositions from a basic set of rules.
- Know the syntax for predicates (including primitive tests and the quantifiers $\forall$ and $\exists$).
- Understand how states for predicates differ from states for propositions.
- Understand how $\vDash$ works for non-quantified predicates (for quantified ones we'll need state updates, which we'll see next time).

C. In Case You Missed The First Class

- The course webpages are at http://cs.iit.edu/~cs536/. Read it carefully for policies and refer to it often to download class notes and homework assignments. Check myIIT→Blackboard for class videos.

D. Formal Proofs of Truth

- For propositions, in addition to semantic truth based on truth tables, there is also a notion of ***provable truth*** based on syntactic manipulation of propositions. E.g., "if $p \wedge q$ is provable then $q \wedge p$ is provable" or "$q \wedge p$ follows from $p \wedge q$".
- ***Notation***: if $\vdash p \wedge q$ then $\vdash q \wedge p$. The $\vdash$ symbol is a "turnstile" (compare to $\vDash$ for semantic truth) and it's pronounced "can prove" or something similar.
- ***Definition***. Given a set of proof rules, two propositions are ***provably equivalent*** if each follows from the other according to those rules. E.g., $p \wedge q$ and $q \wedge p$ are provably equivalent.

Propositional Logic Rules

- You don't need to memorize these rules by name, but you should be able to give the name of a rule. For example, "$(p \rightarrow q) \wedge (p \rightarrow r) \Rightarrow (p \rightarrow r)$ is __________". (Answer: transitivity)
- The rules use the $\Leftrightarrow$ symbol to indicate that each side can be used to prove the other: *lhs*$\Leftrightarrow$*rhs* means that if you can prove the *lhs*, then you can prove the *rhs* and vice versa.

- Using $\vdash$, we can write this as $\vdash lhs \Leftrightarrow rhs$ if and only if both $\vdash lhs$ implies $\vdash rhs$ and $\vdash rhs$ implies $\vdash lhs$.
- ***Logical implication vs logical equivalence***: Analogously to how $(p \leftrightarrow q) \Leftrightarrow T$ lets us know $p \Leftrightarrow q$, if we know $(p \rightarrow q) \Leftrightarrow T$ then we say that $p \Rightarrow q$ (p *logically implies* q).[1] Within the basic proof rules below, $\Rightarrow$ appears in the transitivity rule, and it's critical there: $(p \rightarrow q) \wedge (q \rightarrow r)$ implies $(p \rightarrow r)$ but $(p \rightarrow r)$ doesn't imply $(p \rightarrow q) \wedge (q \rightarrow r)$.

- The set of rules below isn't unique — there are other sets of rules that are equivalent, in the sense of what you can prove using them.

Commutativity $p \vee q \Leftrightarrow q \vee p$

$p \wedge q \Leftrightarrow q \wedge p$ $(p \leftrightarrow q) \Leftrightarrow (q \leftrightarrow p)$

Associativity $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$

$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$

Distributivity/Factoring

$(p \vee q) \wedge r \Leftrightarrow (p \wedge r) \vee (q \wedge r)$

$(p \wedge q) \vee r \Leftrightarrow (p \vee r) \wedge (q \vee r)$

Transitivity [Note: $\Rightarrow$, not $\Leftrightarrow$ here]

$(p \rightarrow q) \wedge (q \rightarrow r) \Rightarrow (p \rightarrow r)$

$(p \leftrightarrow q) \wedge (q \leftrightarrow r) \Rightarrow (p \leftrightarrow r)$

Identity: $p \wedge T \Leftrightarrow p$ and $p \vee F \Leftrightarrow p$

Idempotentcy: $p \vee p \Leftrightarrow p$ *and* $p \wedge p \Leftrightarrow p$

Domination: $p \vee T \Leftrightarrow T$ and $p \wedge F \Leftrightarrow F$

Absurdity: $(F \rightarrow p) \Leftrightarrow T$

Contradiction: $p \wedge \neg p \Leftrightarrow F$

Excluded middle: $p \vee \neg p \Leftrightarrow T$

Double negation: $\neg\neg p \Leftrightarrow p$

DeMorgan's Laws:

$\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$

$\neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$

Definition of $\rightarrow$ $(p \rightarrow q) \Leftrightarrow (\neg p \vee q)$

Definition of $\leftrightarrow$ $(p \leftrightarrow q) \Leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$

Negation of Comparisons (in predicate logic)

$\neg(e_1 \leq e_2) \Leftrightarrow e_1 > e_2$ (similar for $<, >, \geq, =, \neq$)

Substitution: Given p, q, and r, let r' be the result of substituting a q for one or more occurrences of p inside r. If $p \Leftrightarrow q$ then $r \Leftrightarrow r'$. ***Example*** (of substitution): Using $(p \rightarrow q) \Leftrightarrow (\neg p \vee q)$ for r, p for p, and $\neg\neg p$ for q, by substitution, we know $(p \rightarrow \neg\neg p) \Leftrightarrow (\neg p \vee \neg\neg p)$.

E. Sample Proofs

- Proofs in predicate logic give step-by-step reasoning for why we think the truth of one proposition is related to another.
- Here is a proof of $\neg(p \rightarrow q) \Leftrightarrow (p \wedge \neg q)$ (also known as "negation of $\rightarrow$").

$\neg(p \rightarrow q)$	
$\Leftrightarrow \neg(\neg p \vee q)$	Defn $\rightarrow$
$\Leftrightarrow \neg\neg p \wedge \neg q$	DeMorgan's Law
$\Leftrightarrow p \wedge \neg q$	Double negation

[1] Unfortunately, we're running out of word phrases, so "logically implies" in English can mean $\rightarrow$ or $\Rightarrow$. But usually you can figure it out from the context (or just write the symbol, which isn't ambiguous).

- The proof above can be read top-down or bottom-up.
    - Top-down: if $\neg(p \rightarrow q)$ is provable, then $(p \wedge \neg q)$ is provable, using the indicated rule.
    - Bottom-up: if $(p \wedge \neg q)$ is provable, then $\neg(p \rightarrow q)$ is provable, using the indicated rule.
- If you reverse a *lhs* $\Leftrightarrow$ *rhs* proof, you get an equivalent *rhs* $\Leftrightarrow$ *lhs* proof
- So this proof of $p \wedge \neg q \Leftrightarrow \neg(p \rightarrow q)$ is also negation of $\neg$.

$\neg(p \rightarrow q)$	
$\Leftrightarrow\ \neg(\neg p \vee q)$	Defn $\rightarrow$
$\Leftrightarrow\ \neg\neg p \wedge \neg q$	DeMorgan's Law
$p \wedge \neg q$	Double negation

- For another sample proof, here is $((r \rightarrow s) \wedge r) \rightarrow s \Leftrightarrow T$. Its name, "modus ponens" is Latin, but observations of it were known to the ancient Greeks.

$(r \rightarrow s) \wedge r \rightarrow s$	
$\Leftrightarrow\ \neg((r \rightarrow s) \wedge r) \vee s$	Defn of $\rightarrow$
$\Leftrightarrow\ (\neg(r \rightarrow s) \vee \neg r) \vee s$	DeMorgan's Law
$\Leftrightarrow\ ((r \wedge \neg s) \vee \neg r) \vee s$	Negation of $\rightarrow$ [see above]
$\Leftrightarrow\ ((r \vee \neg r) \wedge (\neg s \vee \neg r)) \vee s$	Distribute $\vee$ over $\wedge$
$\Leftrightarrow\ (T \wedge (\neg s \vee \neg r)) \vee s$	Excluded middle
$\Leftrightarrow\ (\neg s \vee \neg r) \vee s$	Identity
$\Leftrightarrow\ T \vee \neg r$	Excluded middle (see below)
$\Leftrightarrow\ T$	Domination

- In contrast, a proof of *lhs* $\Rightarrow$ *rhs* can only be read correctly from the top down.

Avoid Unpleasant Levels of Detail

- In the proof above, if we're being picky with details, then the use of excluded middle to go from $(\neg s \vee \neg r) \vee s$ to $T \vee \neg r$ is

$(\neg s \vee \neg r) \vee s$	
$\Leftrightarrow\ \neg s \vee (\neg r \vee s)$	Associativity of $\vee$
$\Leftrightarrow\ \neg s \vee (s \vee \neg r)$	Commutativity of $\vee$
$\Leftrightarrow\ (\neg s \vee s) \vee \neg r$	Associativity of $\vee$
$\Leftrightarrow\ T \vee \neg r$	Excluded middle

- And if we're being even pickier, we should go from $(\neg s \vee s) \vee \neg r$ to $(s \vee \neg s) \vee \neg s$, since the rule for excluded middle says "$p \vee \neg p \Leftrightarrow T$", not "$\neg p \vee p \Leftrightarrow T$". To avoid this level of detail, let's agree that associativity and commutativity can be used without mentioning them specifically.
- ***Notation***: In propositional logic proofs (and later, predicate logic proofs), we can omit uses of associativity and commutativity rules and treat them as being implicit. (It's still okay to spell them out, of course.)

F. Derived Rules

- If $p \leftrightarrow q$ is a tautology (i.e., $(p \leftrightarrow q) \Leftrightarrow T$), then we can use $p \Leftrightarrow q$ as a ***derived rule***. E.g., to prove modus ponens, we showed $(r \to s) \wedge r \to s \Leftrightarrow T$. Here's an example of using modus ponens. (For r we substitute $p \wedge q$; for s we substitute r.)

$((p \wedge q) \to r) \wedge (p \wedge q) \to r$	Modus ponens
$\Leftrightarrow\ T$	

- Here we see $p \wedge q \to p$ is a tautology, often called "*(left) and-elimination*". (There's also "right and-elimination, which is similar.)

$p \wedge q \to p$	
$\Leftrightarrow\ \neg(p \wedge q) \vee p$	Defn $\to$
$\Leftrightarrow\ (\neg p \vee \neg q) \vee p$	DeMorgan's Law
$\Leftrightarrow\ T \vee \neg q$	Excluded middle
$\Leftrightarrow\ T$	Domination

- Some other common derived rules: [you don't have to memorize these]. Note that *or-introduction* uses $\Rightarrow$, since although p implies $p \vee q$, you can have $p \vee q$ true but with p false. Similarly or-elimination uses $\Rightarrow$ because you can have r true but $p \vee q$ false.
  - *contraposition*: $(p \to q) \Leftrightarrow (\neg q \to \neg p)$
  - *and-introduction*: $p \to (q \to r) \Leftrightarrow p \wedge q \to r$
  - *or-introduction*: $p \ \Rightarrow\ p \vee q$
  - *or-elimination*: $(p \vee q) \wedge (p \to r) \wedge (q \to r) \ \Rightarrow\ r$
  - *not-introduction*: $(p \to F) \Leftrightarrow \neg p$

G. Predicate Logic

- In propositional logic, we assert truths about boolean values; in predicate logic, we assert truths about values from one or more "***domains of discourse***" like the integers.
- We extend propositional logic with *domains* (sets of values), plus variables whose values range over these domains, and operations on values (e.g. addition). E.g., for the integers we add the set $\mathbb{Z}$, operations +, –, *, /, % (*mod*), and relations =, ≠, <, >, ≤, and ≥. We'll also add arrays of integers and array indexing, as in $b[0]$.
- A ***predicate*** is a logical assertion that describes some property of values.

- To describe properties involving values, we add basic relations on values (e.g., less-than). We also have rules over these relations, like $x*0=0$ being a rule of arithmetic.

States for Predicates; Satisfaction and Validity of Predicates (Omitting quantifiers)

- We'll see quantifiers in a bit, but in the meantime, we can still look at the satisfaction relation for predicates.
- ***States with bindings for Domain variables***. With propositions, for states, we've been writing sets of bindings like $\{p=T,\ q=F\}$, where proposition variables are bound to boolean values [2]. With predicates, we can also have bindings like $x=0$, which bind a domain variable to a domain value. E.g., $\{p=T,\ q=F,\ x=0\}$ is a state now. We might have to give the value a name like α if we don't know it precisely, for example $\{p=T,\ q=F,\ x=\alpha\}$. [I'll generally use Greek letters for semantic values.]
- ***States as functions***: Technically, a state is a function from variables (i.e., symbols) to values (proposition variables to boolean values, domain variables to domain values).
- ***Notation***: $\sigma(x)$ is the value of the state function σ on variable x. (I.e., it's the value α where the binding $x=\alpha$ appears in σ).
- ***Notation***: $\sigma(e)$ is the value of the expression e given that its variables take their values from σ. Eg., if $\sigma=\{x=1,\ y=3\}$, then $\sigma(x)=1$, $\sigma(y)=3$, and (picking a random expression), $\sigma(x+y*y)=10$. We'll define the $\sigma(expression)$ idea more formally later, but for now your intuition should be fine.
- ***Notation***: p, q, r, etc. can stand for propositions or predicates. Propositions are also predicates, so technically we don't need to say “propositions or …”, but its good to emphasize the distinction.
- ***Satisfaction of Unquantified Predicates***. State σ ***satisfies*** predicate p, written $\sigma \vDash p$ is defined as follows.
  - $\sigma \vDash p$ (a proposition variable) if $\sigma(p)=T$.
  - $\sigma \vDash e_1=e_2$ holds if the values $\sigma(e_1)$ and $\sigma(e_2)$ are equal. Tests <, ≤, >, ≥, and ≠ are similar.
  - $\sigma \vDash \neg p$ holds if $\sigma \nvDash p$ holds.
  - $\sigma \vDash p \wedge q$ holds if $\sigma \vDash p$ and $\sigma \vDash q$ both hold.
  - $\sigma \vDash p \vee q$ holds if $\sigma \vDash p$ or $\sigma \vDash q$ or both hold.
  - $\sigma \vDash p \rightarrow q$ holds if $\sigma \vDash \neg p \vee q$ holds, or equivalently if $\sigma \vDash \neg p$ or $\sigma \vDash q$ or both hold.
  - $\sigma \vDash p \leftrightarrow q$ holds if $\sigma \vDash (p \rightarrow q) \wedge (q \rightarrow p)$ holds.
- Some points about this definition. First, it seems weird — all we're doing is substituting English words like “and” for logic symbols like “∧”. And that's true. All this means is that the proposition connectives behave as you expect them to if you were to write out things in English.

[2] Remember, $\{p=T,\ q=F\}$ is just more-readable shorthand for $\{(p,T),\ (q,F)\}$.

Second, the last two rules (for implication and biconditional) don't break down the p and q into separate parts, they substitute some larger predicate for $p \rightarrow q$ and $p \leftrightarrow q$. This is because these operations can be described using other connectives.

H. Well-Formed and Proper States

- We have seen well-formedness and properness for states for propositional expressions. States for predicates follow the same concepts, the only difference being the kinds of values that variables map to: In addition to boolean values, there are also values from whatever domain our predicates range over (such as $\mathbb{Z}$ for integers).
    - ***Examples:*** $\{x=1, y=1\}$ is well-formed but $\{x=1, y=x\}$ and $\{x=1, y=x*1\}$ are ill-formed, since they map y to an identifier and an expression respectively. $\{x=1, x=2\}$ is ill-formed because it's not a function (no unique value for x).
    - ***Examples:*** $\{b=3\}$ is proper for $b+2$ but not proper for $b[2]$ (assuming $b[2]$ is an array lookup). It's also improper for $b+x*0$ because it is missing a binding for x (even though the value of x is irrelevant).
- And recall that being well-formed is an intrinsic property of a state; being proper is a relationship between a state and expression or proposition or predicate. Also, when we say "every state" or "no state", we only look at states proper for the context.
    - ***Examples***: We can say "$\sigma(x+0)=\sigma(x)$ in every state" because it's satisfied in every proper state (includes at least a mapping of x to an integer). Ill-formed states like $\{x=y, y=5\}$ or improper states like $\{y=4\}$ are ignored.
- One more point: A state might be proper but that doesn't preclude runtime errors.
    - ***Examples:*** States $\{x=8,\ y=2\}$ and $\{x=8,\ y=0\}$ are both proper for expression x/y, but evaluating the expression gets a runtime error when y is zero.

I. Quantifiers: Syntax

- When a predicate includes a variable, we have to ask for what values of the variable we think the predicate might be true: Some current value? Every value? Some value?
- We use quantifiers to specify all values, some value, and exactly one value.

Universal Quantification

- A ***universally quantified predicate*** (or just "***universal***" for short) has the form $(\forall x \in S.\, p)$ where S is a set and p (the body of the universal) is a predicate involving x. E.g., every integer greater than 1 is less than its own square: $(\forall x \in \mathbb{Z}.\, x > 1 \rightarrow x < x^2)$ [3].
- Often we leave out the set if it is understood. E.g., $(\forall x.\, x > 1 \rightarrow x < x^2)$.

[3] The standard definition of the natural numbers $\mathbb{N}=\{0, 1, 2, \ldots\}$. Avoid any source that says $0 \notin \mathbb{N}$.

Existential Quantification

- An ***existentially quantified predicate*** (or just "***existential***" for short) has the form $(\exists x \in S . p)$ where S is a set and p (the body of the existential) is a predicate involving x. E.g., there is a nonzero integer that equals its own square: $(\exists x \in \mathbb{Z} . x \neq 0 \wedge x = x^2)$.
- Usually the set is understood to be $\mathbb{Z}$ and we leave it out. E.g., $(\exists x . x \neq 0 \wedge x = x^2)$.

Syntactic Equality of Quantified Predicates

- For syntactic equality, which variable you quantify over will make a difference for us, so we'll treat: $(\forall x . x > x - 1) \not\equiv (\forall y . y > y - 1)$. But they'll be logically equivalent: $(\forall x . x > x - 1) \Leftrightarrow (\forall y . y > y - 1)$.

Bounded Quantifiers

- With bounded quantifiers, we abbreviate a quantifier with a condition in the body by moving the condition to the quantifier. We'll take a predicate with bounded quantifier to be ≡ the one without.
- ***Example 1:*** $(\forall x > 1 . x < x^2) \equiv (\forall x . x > 1 \rightarrow x < x^2)$. ("For every $x > 1$, $x < x^2$".)
- ***Example 2:*** $(\exists x \neq 0 . x = x^2)$. $\equiv$ $(\exists x . x \neq 0 \wedge x = x^2)$. ("There's some nonzero x such that $x = x^3$".)
- ***Definition***: A ***bounded quantifier*** takes the form $Q\, p\, .\, q$ as an ≡ abbreviation for $Q\, x\ op\ r$ (where Q is $\forall$ and op is $\rightarrow$ or Q is $\exists$ and op is $\wedge$). More specifically,
- ***Definition***: $\forall p . q$ means $\forall x . p \rightarrow q$ where x appears in p and x is understood to be the variable we are quantifying over.
- ***Definition***: $\exists p . q$ means $\exists x . p \wedge q$ where z appears in p and x is understood to be the variable we are quantifying over. (Note: it's $p \wedge q$ here; compare with $p \rightarrow q$ for bounded universals.)
- It's important to expand bounded $\forall$ and $\exists$ to the correct connectives. For example, take
  - $\exists x \in \mathbb{Z} . x > 1 \wedge x = x^2$ which is false
  - $\exists x \in \mathbb{Z} . x > 1 \rightarrow x = x^2$ which is true, e.g. when $x = -1$.

Parentheses for Quantified Predicates

- We'll treat $\forall$ and $\exists$ as having low precedence. (Note: Some people use high precedence). So the body of a quantified predicate is as long as possible.
- ***Example 3***. $\forall x \in \mathbb{Z} . x > 1 \rightarrow x < x^2$ means $(\forall x \in \mathbb{Z} . ((x > 1) \rightarrow (x < x^2)))$.
- ***Example 4***: $\forall x \in \mathbb{Z} . \exists y \in \mathbb{Z} . y \leq x^2$ means $(\forall x \in \mathbb{Z} . (\exists y \in \mathbb{Z} . (y \leq x^2)))$.
- ***Notation***: Q means $\forall$ or $\exists$.
- If we have (… $Q\, x$ …) where the two parentheses shown match, then the body can't extend past the right parenthesis, and we get (… $Q\, x .$ (…)).
- ***Example 5***: $(\exists y \in \mathbb{Z} . y > 0 \wedge x > y) \rightarrow x \geq 1 \equiv ((\exists y \in \mathbb{Z} . ((y > 0) \wedge (x > y))) \rightarrow (x \geq 1))$
- For full parenthesizations, we add parentheses around basic tests, but we still omit them around variables and constants. Let's also omit them around array indexes, so we'll write $(b[x+1] > y)$, not $(b[(x+1)] > y)$.

- **Example 6**: $x>0 \wedge y\leq 0$ expands to $((x>0)\wedge(y\leq 0))$.

J. Quantifiers: Semantics

DeMorgan's Laws For Quantified Predicates

- For quantified predicates, there are two more ***DeMorgan's Laws***:
  - $(\neg\forall x.p) \Leftrightarrow (\exists x.\neg p)$ and $(\neg\exists x.p) \Leftrightarrow (\forall x.\neg p)$
- With bounded quantifiers, because of how $\rightarrow$, $\neg$, and $\wedge$ are related,
  - $(\neg\forall p.q) \Leftrightarrow (\exists p.\neg q)$. I.e., $(\neg\forall x.p\rightarrow q) \Leftrightarrow (\exists x.\neg(p\rightarrow q)) \Leftrightarrow (\exists x.p\wedge\neg q) \Leftrightarrow (\exists p.\neg q)$.
  - $(\neg\exists p.q) \Leftrightarrow (\forall p.\neg q)$. I.e., $(\neg\exists x.p\wedge q) \Leftrightarrow (\forall x.\neg(p\wedge q))$
    $\Leftrightarrow (\forall x.\neg p\vee\neg q) \Leftrightarrow (\forall x.p\rightarrow\neg q) \Leftrightarrow (\forall p.\neg q)$.
- ***Example 7***: $\neg(\forall x.x>0)\Leftrightarrow(\exists x.\neg(x>0)) \Leftrightarrow (\exists x.x\leq 0)$
- ***Example 8***: $\neg(\forall x>0.x^2=x)\Leftrightarrow(\exists x>0.x^2\neq x)$
- ***Example 9***: $\neg(\exists x.x\leq 0 \wedge x>0) \Leftrightarrow (\forall x.\neg(x\leq 0 \wedge x>0)) \Leftrightarrow (\forall x.x>0 \vee x\leq 0)$.

Proofs of Quantified Predicates

- Formal systems for proving predicates are pretty complicated; rather than study one of them, let's rely on an informal idea of how to prove universally and existentially quantified predicates.
- In general, to prove $\forall x.p$, you prove p but without imposing any restrictions on x. If you need to restrict x, then this needs to be part of the body of the quantified predicate.
- ***Example 10***: To prove $\forall x\in\mathbb{Z}.x\neq 0 x\leq x^2$, we can say "Let x be an integer. Assume that x isn't zero. In that case, $x\leq x^2$."
- To prove $\exists x\in S.p$, you name a ***witness value*** for x and prove p holds if x has that value.
- ***Example 11***: To prove $\exists x\in\mathbb{Z}.x\neq 0 \wedge x\geq x^2$, the only value that works as a witness is 1. More generally, there may be multiple witness values that work; we just need to name one.
- If a predicate includes unquantified variables, then for it to be a tautology, it has to hold for all possible values of those quantified variables. It's a contradiction if it fails for all values, and it's a contingency if it holds for some values but not some others. (I.e., unquantified variables are "implicitly universally quantified".)
- ***Example 12***: $x>0\rightarrow\exists y.y^2<x$ is a tautology because $\forall x.(x>0\rightarrow\exists y.y^2<x)$ holds.
- ***Example 13***: $x>0\rightarrow y^2<x$ is a contingency because it holds for some x and y (like $x=2$ and $y x=1$) but fails for others (like $x=y=1$).
- ***Example 14***: $\exists y . (y<0 \wedge y>x^2)$ is a contradiction because it fails for every value of x.

K. Predicate Functions

- Often, we'll give names to predicates and parameterize them. In programming languages, these predicate functions are written as functions that yield a boolean result.

- ***Example 15***: we might define $even(x) \equiv (x\%2)=0$, where % is the remainder operator. E.g., $Even(3) \equiv (3\%2)=0 \Leftrightarrow 1=0 \Leftrightarrow F$.
- In a programming language, the body of a predicate function can be a general program — one that uses loops and decisions. We want our predicates to be simpler than that: We're going to use predicates to augment our programs with specifications, and it won't help if debugging a predicate function body is exactly as hard as debugging a general program.
- So we'll restrict ourselves to predicate functions that take one or more parameter variables and evaluate a predicate on those variables. ***Example***: $p(x) \equiv x>y \wedge x<z$.
- The body of the predicate function is (surprise) a predicate that evaluates to true or false, not an expression that yields a number.
- Function: $sqrt(x) \equiv$ (in pseudocode) r where $r=0$ if $x \le 0$ and where $r*r=x$ if $x \ge 0$.
- Predicate function: $isSqrt(x,r) \equiv (x \le 0 \wedge r=0) \vee r*r=x$.
- The body of a predicate function can use the parameter variables and the built-in relations for our datatypes (for integers, <, ≤, etc.) along with the propositional connectives (∧, ∨, etc.)
- ***Example 16***: Let's define $IsZero(b, m)$ to be true if the first m elements of b are all zero. To help, let's assume $size(b)$ gives the number of elements in b.
- Rewriting, $IsZero(b,m)$ means that $b[0], b[1], \ldots, b[m-1]$ all equal 0.
- It might be tempting to write $IsZero(b,m) \equiv b[0]=0 \wedge b[1]=0 \wedge \ldots \wedge b[m-1]=0$
- But the right hand side is not a predicate; a predicate needs a fixed number of conjuncts being and'ed together.
- To write this as a predicate, we look for a pattern in our informal description: "$b[0], b[1], \ldots, b[m-1]$ all $=0$" is equivalent to "$b[i]=0$ for (every) $i=0, 1, \ldots, m-1$". The implied "every" i tells us we need a universal quantifier $\forall$.
- So we can get $IsZero(b,m) \equiv \forall i. 0 \le i<m \rightarrow b[i]=0$. With bounded quantifiers, we can write $IsZero(b,m) \equiv 0 \le i<m . b[i]=0$.
- Another way to look at a description and find an equivalent predicate is to imagine writing a loop to calculate whether the property is true or false.
- E.g., with "*b[0], b[1], …, b[m-1]* all *= 0*" we might imagine a loop

```
for i = 0 to m-1
    if b[i] ≠ 0 then return false
return true
```

- The "*for* $i=0$ *to* $m-1$" tells us we need to search for i in the range $0 \le i<m$.
- The loop returns true only if ***all*** the $b[i]$ pass the $=0$ test; this tells us we need $\forall i$.

The general translation for a universal is ∀ *loop var* . ((*var* in search range)→(test on *var*)). For this example, the search range is $0 \le i<m$ and the test on i is $b[i]=0$. This gives us $\forall i. 0 \le i<m \rightarrow b[i]=0$.

- We need a $\exists$ search if our loop needs to return true as soon as it finds a $b[i]$ that passes the test. E.g., if the property had been "At least one of $b[0], b[1], \ldots, b[m-1] = 0$", we might imagine a loop

```
for i = 0 to m-1
    if b[i] = 0 then return true
return false
```

- For an existential we translate the loop to $\exists$ *loop var* . ((*var* in search range) $\wedge$ (test on *var*)). For this example, we get $\exists i.\, 0 \le i < m \wedge b[i] = 0$.
- ***Example 17***: Define $SortedUp(b, m, n)$ so that it is true when array b is sorted $\le$ on the segment $m..n$. As an example, if $b[0..3]$ is $[1, 3, 5, 2]$, then $SortedUp(b, 0, 2)$ is true because $1 \le 3$ and $3 \le 5$ but $SortedUp(b, 0, 3)$ is false because we don't have ($1 \le 3$ and $3 \le 5$ and $5 \le 2$).
  - Another way to describe $SortedUp(b, m, n)$ is that each element in the list $b[m], b[m+1]$, …, $b[n-2], b[n-1]$ is ≤ the element to its right.
  - Or expanding further, $b[m] \le b[m+1], b[m+1] \le b[m+2]$, …, $b[n-1] \le b[n]$. We can generalize this to $b[i] \le b[i+1]$ for $i = m, m+1, m+2, \ldots, n-1$. To get a formal predicate, we need a $\forall$ over i:
  - $SortedUp(b, m, n) \equiv \forall i.\, m \le i < n \rightarrow b[i] \le b[i+1]$. We can hoist the parts of this that don't depend on the quantified variable i:
    - $SortedUp(b, m, n) \equiv \forall m \le i < n.\, b[i] \le b[i+1]$.
  - If we want to make sure that the indexes are legal, instead of $m \le i < n$ we can make sure $0 \le m < n < size(b)$ and write $0 \le m \le i < n < size(b)$
  - Note: Different generalizations of a property can lead us to different predicates.
  - If we generalize

    $b[m] \le b[m+1], b[m+1] \le b[m+2]$, …, and $b[n-1] \le b[n]$

    to $b[m+j] \le b[m+j+1]$ for $j = 0, 1, \ldots, n-1-m$

    we get $\forall 0 \le j < n-m\ .\, b[m+j] \le b[m+j+1]$ (and $0 \le m \le n < size(b)$).
- ***Example 18***: Let's find a definition for $Extends(b, b')$ so that it's true if b' is an extension of b. I.e., $b[0] = b'[0], b[1] = b'[1]$, … for all elements of b.
  - Note b' can be the same length as b or can be longer.
  - E.g., if b is $[1, 6, 2]$ and b' is $[1, 6, 2, 8]$, then $Extends(b, b')$ is true and $Extends(b', b)$ is false.
  - Here's one solution: $Extends(b, b') \equiv size(b) \le size(b') \wedge \forall 0 \le k < size(b).\, b[k] = b'[k]$.