CS109A Notes for Lecture 1/10/96

Major Theme: Data Models

- Data model = A way of representing (some kinds of) information in a computer.
  - ☐ *Static part*: represents the information.
  - ☐ *Dynamic part*: operations on the information.
- Section 1.3 discusses examples: lists, trees, logic, use of logic to design switching circuits.

Example: The *set* is another common, important data model.

- Static part: Sets are characterized by a membership concept. Sets have *members.* $S = \{a, b, c\}$ says that the members of set S are the elements a, b, and c.
- Dynamic part: Many operations are used. Examples:
  - ☐ $insert(x, S)$ adds element x to the members of set S.
  - ☐ $union(S, T)$ produces the union of sets S and T.

Example: Programming languages like C have a data model. The C model is discussed in Section 1.4.

- The static part of the C model is the language's *type system.* Key elements include:
  - ☐ Basis = atomic types, e.g. char, int, enumerations.
  - ☐ Inductive part = *type constructors* = ways to build new types and their values, e.g. array-formation, struct formation, pointers.
- The dynamic part consists of ways to operate on values:

- ☐ Operations, e.g. arithmetic such as $+$, logical such as &&, comparison such as $<$, assignment ($=$).

- ☐ Structure-access operations, e.g., `->`.

- ☐ Creation/destruction operations such as `malloc` and `free`.

Major Theme: Recursion

Express a concept, algorithm, proof, etc. in terms of smaller instances of the same thing.

Example: To add two n-digit numbers, start by assuming there is a carry into the low-order position.

- *Basis case*: If $n = 0$, just produce the carry.

- *Inductive case*: Add the low-order digits plus the carry-in, generating a carry into the next place (which may be 0). Then recursively add the high-order $n-1$ digits with the new carry.

Propositional Logic

- *Constants*: `TRUE` and `FALSE` (often written 1 and 0, respectively).

- *Propositional variable* = symbol that represents the truth or falsehood of a "proposition," i.e., a statement about something.

  - ☐ Examples are propositional variable p standing for "it is raining" or variable q standing for "$X < Y + Z$."

Propositional Logic Expressions

Built from operands (constants and variables) and *logical operators*, which are functions with Boolean arguments and result.

Most common operators:

a) `AND`, `OR`, `NOT`: the usual stuff as in `if(...)`.

b) *Implies.* $p \rightarrow q$ has value `TRUE` unless p is `TRUE` and q is `FALSE`.

- ☐ When p is false, we say that $p \to q$ is *trivially true.*
- ☐ e.g.: "if $2+2=5$ then the moon is made of cheese."

c) *Equivalence* or "if and only if." $p \equiv q$ is true if p and q are both true or both false. It is false if exactly one of p and q is true.

Predicates and Atomic Formulas

Atomic formula = propositional variable (called a *predicate*) with arguments, e.g., $p(X, Y)$.

- True or false depending on what X and Y are.

Example: Suppose arguments of p were integers, and $p(X, Y)$ is assumed to mean $X^2 > Y$. Then $p(2,3)$ is true, but $p(-2,5)$ is false.

- Expressions can be built from atomic formulas instead of propositional variables.

Example: $p(X) \to q(X)$ = "if p is true about some object X, then q is also true about X."

- If there is no X for which $p(X)$ is true, then $p(X) \to q(X)$ is said to be true *vacuously.*
- e.g.: "every green elephant wears boxer shorts."

Quantifiers

The symbol $(\forall X)$ stands for "for all X," while $(\exists X)$ stands for "there exists at least one X."

- Quantifiers are expressed variously in English.
- And a global $(\forall X)$ is often expressed without any equivalent to "for all." X just appears in the statement.

Example: Here are some ways that "for all P, if P is a prime and $P > 2$ then P is odd" could be expressed:

1. Use "every": "every prime $P > 2$ is odd."
2. Use "each": "each P bigger than 2 that is a prime is odd."

3. Use nothing: "if P is a prime and $P > 2$ then P is odd."

Class Problem for Next Time

Teaching CS145 on database systems last quarter, I made the following definition; never mind if the terms sound mysterious:

> "If relation R is in Boyce-Codd Normal Form, then for every nontrivial functional dependency $X \rightarrow Y$, X is a superkey."†

Later on that day, I used what I thought was the above definition in the following way:

> "If $X \rightarrow Y$ is a nontrivial functional dependency but X is not a superkey, then R is not in Boyce-Codd Normal Form."

Question: Did my second statement follow from the first? Why or why not?

- *Hint*: We might be tempted to see this problem as one of predicate logic, with R, X, and Y as variables. However, to make things simpler, let's focus on a particular R, X, and Y. Then we can think of three propositional variables:

  1. p: "R is in Boyce-Codd Normal Form."
  2. q: "$X \rightarrow Y$ is a functional dependency."
  3. r: "X is a superkey."

- If you solve this problem for propositions as above, try formulating the same question in predicate logic and solving it.

† Note that the $\rightarrow$ symbol for functional dependencies has nothing at all to do with the same logical symbol meaning "implies."