FORMAL SPECIFICATION TECHNIQUES
FORMAL SPECIFICATION TECHNIQUES

Robert B. France

france@cse.fau.edu

Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523

Tel: 970-491-6356
Fax: 970-491-2466

Types of properties that can be investigated with FSTs

What are some of the properties that can be analyzed with FSTs?

- Correctness: establishing that a model satisfies its specification.
- Safety: establishing that a model/specification does not allow undesired behavior.
- Liveness: establishing that a model/specification will eventually do certain desired things.
- Security: establishing that unauthorized accesses to sensitive data and programs are not permitted by model.
- Timeliness: establishing that modeled behavior is consistent with timing constraints.

FSTs vs. testing/prototyping
Prototyping and testing are currently the principle methods for exploring designs and validating implementations; these are expensive, and do not cover the range of behaviors that a piece of software may exhibit.
Limitations of formal modeling of software

- Formal specifications:
  - can say too little or too much;
  - can utilize inappropriate abstractions;
  - can be wrong!

- Analysis can be faulty.

- A formal model is at best an approximation of the real world; mathematical models cannot account for real-world behavior with perfect accuracy.

- Formal models make assumptions about reality; if these are violated then behavior cannot be predicted using the model.

- Formal proofs do not correspond to evidence needed to convince human reviewers.
Formal specification myths

- FSTs eliminate the need for testing.
- FSTs eliminate the need for natural language descriptions.
- You need a PhD in mathematics to understand FSTs.
- Use of FSTs preclude the use of other methods.
- FSTs guarantee correctness.
- FSTs are concerned only with proving programs correct.
- Only highly critical systems benefit from FSTs.
- FSTs increase cost of development.
- FSTs are time-consuming.
- FSTs lack tools.
- FSTs stifle creativity.
Levels of Rigor

**Level 0:** No use of FSTs.

**Level 1:** Use of concepts and notation from discrete maths.

**Level 2:** Use of formalized specification languages with some supporting tools (e.g., typecheckers, context-sensitive editors).

**Level 3:** Use of formal specification languages with supporting environments, including animators, and mechanized theorem proving and proof checking tools.
Example of a Level 0 specification

Calculate the mean average of a set of employee salaries.

1. Read in the list, L, of employee salaries from the Employee file.
2. Sum the values in L in Total.
3. Divide Total by the number of elements in L and output the result.
Example of a Level 1 specification

Calculate the mean average of a set of employee salaries.

**input** empsals: list of \( \mathbb{N} \) (\( \mathbb{N} \) is the set of natural numbers)  
**output** avg : \( \mathbb{R} \) (\( \mathbb{R} \) is the set of real numbers)

**Pre-Condition**  
\[ \text{empsals} \neq \text{emptylist} \]

**Post-Condition**  
\[ avg = \left( \sum_{i=1}^{\#(empsals)} \text{empsals}(i) \right) \text{ div } \#(empsals) \]
Example of a Level 2 specification

A Z specification of the calculate average salaries problem (average in this case is formed by carrying out an integer division).

\[
\begin{align*}
\text{sum} : &\quad \text{seq } \mathbb{N} \rightarrow \mathbb{N} \\
(\text{sum}(\emptyset) = 0) &\quad \forall l : \text{seq } \mathbb{N} \cdot \text{sum}(l) = \text{head}(l) + \text{sum}(\text{tail}(l))
\end{align*}
\]

\[
\begin{align*}
\text{average} &\quad \text{empsals?} : \text{seq } \mathbb{N} \\
\text{avg!} : &\quad \mathbb{N} \\
\text{avg!} = &\quad \text{sum(empsals?) div } \#(\text{empsals})
\end{align*}
\]
Choosing appropriate levels of rigor

Higher levels are not always superior to lower levels.

- If FSTs are to be used for documentation only level 1 is appropriate.
- If FSTs are to be used for justifying designs of critical systems then level 3 may be preferred.
- If FSTs are to be used to help gain insight into a problem and/or solution level 2 may be sufficient.
- Level 0 can be used for routine (well-understood) applications.
Logic is concerned with mechanized reasoning.

Reasoning, as used here, involves determining the ‘truth’ of some statement (the conclusion) based on assumed ‘truths’ of other statements (the premises).

Logic is used to determine whether the truth of the conclusion follows necessarily from truth of preceding statements.

Logic is used to analyze the form of arguments, not to determine the merits of an argument’s content.
Form vs Content

If the program syntax is faulty or if program execution results in division by 0, then an error message will be generated. Therefore, if the computer does not generate an error message, then the program syntax is correct and its execution does not result in division by 0.

Form:

If \( p \) or \( q \), then \( r \).
Therefore, if not \( r \), then not \( p \) and not \( q \).
Propositional logic

A proposition is a statement that is true or false but not both.

Examples:
- The A system is in shut-down mode.
- The Employee file has been created.

The following are not propositions:
- He is a student.
- You will be a logician at the end of this course.
Boolean expressions

Boolean expressions are constructed from the constants true and false, boolean variables (e.g., p, q, r, s), and boolean operators (e.g., $\leftrightarrow, \lor, \land, \Rightarrow$).

Boolean expressions are used to represent the form of simple and compound propositions.

Example: If the mixer is activated and the inlet valve is open, the reactor is working normally.

p : the mixer is activated
q : the inlet valve is open
r : the reactor is working normally
Form: $(p \land q) \Rightarrow r$
Model-based semantics for propositional logic

A state is an assignment of truth values (true, false) to boolean variables.

The value of a boolean expression in a state is determined by truth tables.

A boolean expression is satisfied in a state if its value is true in that state.

A boolean expression is satisfiable if there is a state in which it is satisfied.

A boolean expression is valid if it is satisfied in every state. A valid boolean expression is called a tautology.
Propositional logic cannot be used to reason about classes of objects.

For example, from the statements,

*All monitoring computers are ready.*
*X12 is a monitoring computer.*

One cannot infer that

*X12 is ready.*

using propositional logic.

Predicate logic provides notation for expressing properties involving classes of objects, such as:

- Cubes of integers are never even.
- Some integer is a prime number.
- A student receives a grade for every course in which they enroll.
Predicates, functions, and formulas

A function maps objects to other objects.
Example: $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$
The above is called the signature of a function.

A predicate is a function that returns a true or false value.
Example: $\leq : \mathbb{N} \times \mathbb{N} \to boolean$

Predicate logic formulas include boolean expressions, and formulas of the form:

- $\forall a : A \bullet P$ (unrestricted universal quantifier)
- $\forall a : A \mid R \bullet P$ (restricted universal quantifier)
- $\exists a : A \bullet P$ (unrestricted existential quantifier)
- $\exists a : A \mid R \bullet P$ (restricted existential quantifier)
A small example

Every senior must take one maths course and pass one programming course to graduate.

\[ \forall s : \text{student} \mid \text{senior}(s) \bullet \text{graduate}(s) \Rightarrow \\
(\exists c1, c2 : \text{course} \mid \text{math}(c1) \land \text{prog}(c2) \bullet \text{take}(s, c1) \land \text{pass}(s, c2)) \]