Integrating Formal Specification and Verification Techniques into the Software Engineering Process

After-The-Fact-Verification

- System is built using standard software development approach
- Formal specification is written after the system development is completed
- Properties are proved about the specification and/or code

Requirements
  Preliminary Design
  Detailed Design
  Code
  Formal Specification

After-The-Fact-Verification

- Disadvantages
  - Additional 30-50% cost for the verification effort
  - Additional time occurs after the product development is completed
  - Errors are more costly to fix because the development cycle is completed

Parallel Verification Effort

- Two teams
  - Software development team
  - Specification and verification team
- Requires constant communication between the two teams
- Developed code is verified to be consistent with the formal specification
Parallel Verification Effort

- Disadvantages
  - Insufficient communication results in a large gap between the code and the formal specification
  - Costs the same as for the after-the-fact approach

Fully Integrated Approach

- Formal methods integrated into the software development process
- Software development team and the formal verification team are one
- Software developers use formal specifications as their design notation

Fully Integrated Approach

- Advantages
  - Can reason rigorously about the design before writing any code
  - Costs less than with the other two approaches
  - Errors are likely to be found earlier in the software development process

ASLAN
A Formal Specification Language and Formal Verification System
Models
Access Control
Considers Subjects and Objects
Requirements:
1) If subject S has read access to object O, then
   \( \text{security\_level}(S) \geq \text{security\_level}(O) \)
2) If subject S has write access to object O, then
   \( \text{security\_level}(S) \leq \text{security\_level}(O) \)

Formal Specification
State Machine
Relates values of variables before and after each state transition

\[
\text{Exchange}(X,Y) \\
\text{New\_Value}(X) = Y \\
& \text{New\_Value}(Y) = X
\]

Formal Specifications
Algebraic
Relates results of sequence of operations

\[
\text{Exchange(Exchange}(X,Y)) = (X,Y) \\
\text{First(Exchange}(X,Y)) = \text{Last}(X,Y) \\
\text{Last(Exchange}(X,Y)) = \text{First}(X,Y)
\]

Design Verification
• Consistency between the model and the specification
• Assumes
  – Model is appropriate
  – Specification is complete
Code Verification

- Consistency between the specification and the implementation
- Assumes
  - Specification is complete
  - Implementation language is correctly defined

Aslan

- An integrated methodology for design, specification, implementation, and verification of software
- Enforces the establishment of rigorous connections between successive stages of development
  - Identification and modeling of critical requirements
  - Design specifications
  - Verification of specifications
  - Program design specifications
  - Verification of implementations

Components of Aslan

- Aslan specification language
- Aslan specification processor
- Interactive theorem prover
- Verification condition generator (vcg)

Aslan Language

- State machine representation
- Non-procedural
- Assertion language: extension of first-order predicate calculus
- Language elements
  - Types
  - Constants
  - Variables
  - Definitions
  - Initial conditions
  - Invariants
  - Constraints
  - Transitions
  - Levels
  - Implementations (Mappings)

Type

- User,
- Book,
- Mystery Subtype Book

Type

- Status is (In, Out, Lost),
- Book_Collection is set of Book

Type

- Pos_Integer is Typedef i:Integer (i>0)
Constant Book_Limit: Pos_Integer,
Title(Book): Book_Title

Variable Library:Book_Collection,
Checked_Out(Book): Boolean

Define Copy_Of(b1,b2:Book): Boolean ==
  Author(b1) = Author(b2)
  & Title(b1) = Title(b2)

Axiom Forall b1,b2,b3:Book(
  Left_Of(b1,b2) & Left_Of(b2,b3)
  → Left_Of(b1,b3))

Initial
Library = Empty
& Forall u:User (Number_Books(u) = 0)
& Forall b:Book (~Checked_Out(b))

Invariant
Forall u:User(
  Number_Books(u) <= Book_Limit)
& Forall u:User,b1,b2:book(
  Checked_Out_To(u,b1)
  & Checked_Out_To(u,b2)
  & Copy_Of(b1,b2)
  → b1 = b2)

Constraint
Time > Time' | (Time=0 & Time' > 0)

Critical Requirements

Invariant is about a single state

Constraint is across successive states
Transition Return(b:Book)
Entry
    Checked_Out(b)
Exit
Forall b1:Book(
    if b1=b then ~Checked_Out(b1)
    else Nochange(Checked_Out(b1))
fi)

Aslan Procedural Operators

• Becomes Operator
• Conditional Statements
• Alternative Statements

Transition Check_Out(u:User, b:Book)
Entry
    Available(b)
    & Number_Books(u) < Book_Limit
    & Forall b1:Book (Checked_Out_To(u,b1) → -Copy_Of(b,b1))
Exit
    Number_Books(u) Becomes Number_Books'(u) +1
    & Checked_Out(b) Becomes True
    & Responsible(b) Becomes u
    & Never_Out(b) Becomes False

Conditional Treated Like a Procedural Statement

Transition Login(u:user, p:password)
Exit
if Password_OK(u,p)
    then
        Logged_In(u) Becomes true
        & Nochange(Failed_Logins)
    else
        Failed_Logins = Failed_Logins' + 1
        & Nochange(Logged_In)
fi
Transition Login(u: user, p: password)
Exit
  Password_OK(u, p)
& Logged_In(u) Becomes true
& Nochange(Failed_Logins)
  ~Password_OK(u, p)
& Failed_Logins = Failed_Logins' + 1
& Nochange(Logged_In)

Alternative Treated Like a Procedural Statement
Transition Login(u: user, p: password)
Exit
  Password_OK(u, p)
& Logged_In(u) Becomes true
ALT
  ~Password_OK(u, p)
& Failed_Logins = Failed_Logins' + 1

Implementation Mappings
All types, constants, variables and transitions are mapped to the next lower level

Aslan Specification Processor
• Reads Aslan specifications including the critical requirements and implementation mappings
• Generates proof obligations

Critical Requirements are Invariants and Constraints

Invariants must hold in every reachable state

Constraints must hold between two successive states

Top Level Proof Obligations
• Initial conditions satisfy the invariants
• Each transition preserves the invariants and satisfies the constraints
Initial Conditions Satisfy the Invariants

Initial_Assertion $\rightarrow$ Invariant

Each Transition Preserves the Invariants and Satisfies the Constraints

For the Normal Case:

Invariant' & Entry' & Exit $\rightarrow$ Invariant & Constraint

For Each Exception:

Invariant' & Except' & Exit $\rightarrow$ Invariant & Constraint

Refinement Proof Obligations

- Lower level initial conditions implement the higher level initial conditions
- Lower level transitions correctly implement the corresponding higher level transitions with respect to the implementation mapping
- Lower level transitions that do not correspond to a higher level transition preserve a refinement of the higher level invariants and satisfy a refinement of the higher level constraints

Implementation Mapping

Lower level initial conditions imply a mapping of the higher level initial conditions and the lower level invariants

Initial_Low $\rightarrow$

IMPL(Initial_High) & Invariant_Low
Unmapped lower level transitions preserve invariants and satisfy the constraints

For the normal case:
\[ \text{IMPL(Invariant}_{\text{High}}') \& \text{Invariant}_{\text{Low}}' \]
& Entry' & Exit
\[ \rightarrow \]
\[ \text{IMPL(Invariant}_{\text{High}}) \& \text{Invariant}_{\text{Low}} \]
& IMPL(Constraint_{\text{High}})
& Constraint_{\text{Low}}

For each exception:
\[ \text{IMPL(Invariant}_{\text{High}}') \& \text{Invariant}_{\text{Low}}' \]
& Except-i' & Exit-i
\[ \rightarrow \]
\[ \text{IMPL(Invariant}_{\text{High}}) \& \text{Invariant}_{\text{Low}} \]
& IMPL(Constraint_{\text{High}})
& Constraint_{\text{Low}}

For each refined transition
Assuming the mapping
\[ T_{\text{High}} == T_{\text{Low}} \]
For the normal case
Correct application:
\[ \text{IMPL(Entry}_{\text{High}}) \& \text{IMPL(Invariant}_{\text{High}}) \]
& Invariant_{\text{Low}}
\[ \rightarrow \]
Entry_{\text{Low}}

Correct refinement:
\[ \text{IMPL(Entry}_{\text{High}}') \& \text{IMPL(Invariant}_{\text{High}}') \]
& Invariant_{\text{Low}}' & Exit_Low
\[ \rightarrow \]
\[ \text{IMPL(Exit}_{\text{High}}) \& \text{Invariant}_{\text{Low}} \]
& Constraint_{\text{Low}}

For each exception
Correct application:
\[ \text{IMPL(Except-i}_{\text{High}}) \& \text{IMPL(Invariant}_{\text{High}}) \]
& Invariant_{\text{Low}}
\[ \rightarrow \]
Except-i_{\text{Low}}

Correct refinement:
\[ \text{IMPL(Except-i}_{\text{High}}') \& \text{IMPL(Invariant}_{\text{High}}') \]
& Invariant_{\text{Low}}' & Exit-i_{\text{Low}}
\[ \rightarrow \]
\[ \text{IMPL(Exit-i}_{\text{High}}) \& \text{Invariant}_{\text{Low}} \]
& Constraint_{\text{Low}}
Each Entry(Except)/Exit pair can have its own mapping and the mapping expressions may be arbitrarily complex.

Consider
\[
T_{\text{High}.i} \equiv \begin{cases} 
\text{expr} & \text{then } T1.j \\
\text{else} & T2.k
\end{cases}
\]

This is equivalent to

\[
T_{\text{High}.i} \equiv (\text{expr} \land T1.j) \lor (\neg \text{expr} \land T2.k)
\]

Four proof obligations are produced.

For correct application:

\[
\begin{align*}
\text{IMPL} & (\text{Except-i\_High}) & \land & \text{IMPL}(\text{Invariant\_High}) \\
& \land & \text{Invariant\_Low} & \land \text{expr} \\
\rightarrow & & \text{Except-j\_T1}
\end{align*}
\]

\[
\begin{align*}
\text{IMPL} & (\text{Except-i\_High}) & \land & \text{IMPL}(\text{Invariant\_High}) \\
& \land & \text{Invariant\_Low} & \land \neg \text{expr} \\
\rightarrow & & \text{Except-k\_T2}
\end{align*}
\]

For Correct refinement:

\[
\begin{align*}
\text{IMPL} & (\text{Except-i\_High}') & \land & \text{IMPL}(\text{Invariant\_High}') \\
& \land & \text{Invariant\_Low'} & \land \text{expr'} & \land \text{Exit-j\_T1} \\
\rightarrow & & \text{IMPL}(\text{Exit-i\_High}) & \land & \text{Invariant\_Low} & \land \text{Constraint\_Low}
\end{align*}
\]

\[
\begin{align*}
\text{IMPL} & (\text{Except-i\_High}') & \land & \text{IMPL}(\text{Invariant\_High}') \\
& \land & \text{Invariant\_Low'} & \land \neg \text{expr'} & \land \text{Exit-k\_T2} \\
\rightarrow & & \text{IMPL}(\text{Exit-i\_High}) & \land & \text{Invariant\_Low} & \land \text{Constraint\_Low}
\end{align*}
\]