Software Engineering

(5) Formal Methods

Sokendai / National Institute of Informatics
Fuyuki Ishikawa / 石川　冬樹
f-ishikawa@nii.ac.jp / @fyufyu
http://research.nii.ac.jp/~f-ishikawa/

大学共同利用機関法人 情報･システム研究機構
国立情報学研究所
National Institute of Informatics

TOC

V&V

- Verification（検証）

  Are you building the things right?

  - Given criteria about correctness（正当性）？
  - Often considered for each phase and each deliverable

- Validation（妥当性確認）

  Are you building the right things?

  - Given criteria about Validity（妥当性）
  - Basically for the whole product/service

➡ Called V&V to refer to the whole activities

f-ishikawa@Sokendai

V&V: Positioning

- Validation makes questions on the ultimate goals of customers and users
  - We conduct acceptance testing (next week) and questionnaires but there will always be uncertainty and continuous effort is required
- Verification makes questions on (sub-)objectives necessary for validity
  - Most of formal methods (this week) and testing (next week) work on verification but contribute to validation as well

TOC

- V&V
- <u>Formal Methods</u>
- Underlying Theories
- Example Methods

Rigor and Expressiveness of Models

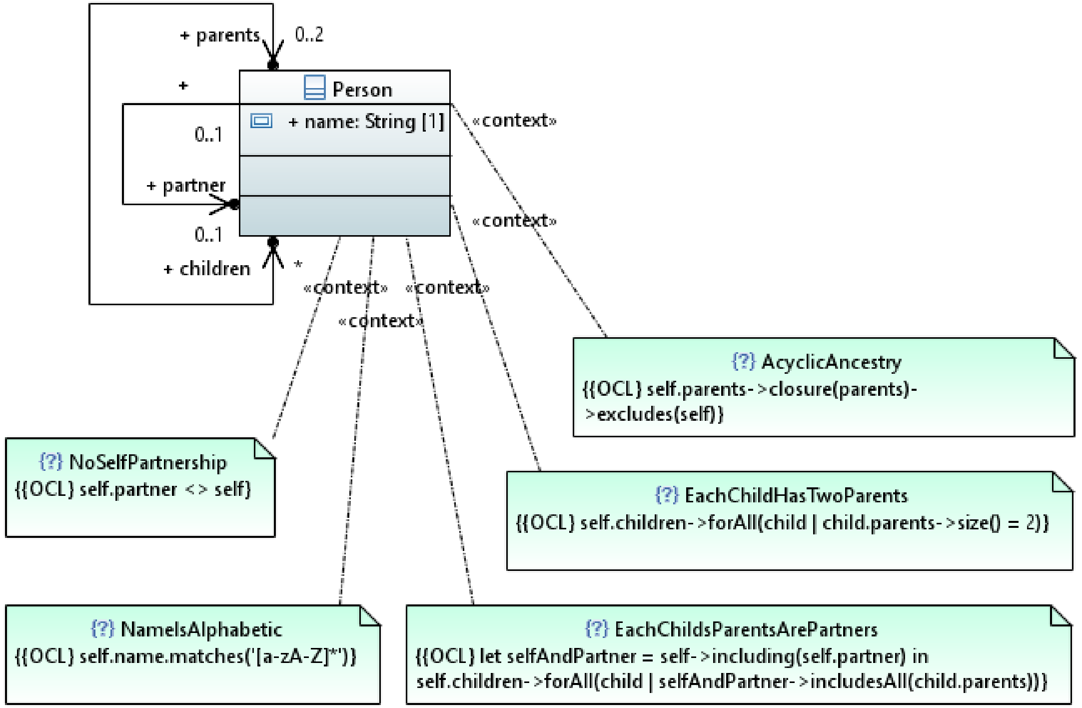
- Diagram-based models are sometimes like “sketch”
  - Syntax of descriptions (including diagrams) is usually strict
  - Semantics is sometimes vague (e.g., in old UML versions)
  - ➡ People may have different interpretations, e.g., if they implement interpreters for state transitions
  - The amount of information inside the model is small
  - ➡ We typically have operations signatures (types of inputs/outputs)
- Natural language models (documents) are more difficult
  - Too many points to check, possibly unstructured, …

Formal Methods

- Formal Methods（形式手法・フォーマルメソッド）
  - Refers to a variety of approaches based on mathematical logic for efficient development of high-quality software systems
  - Makes use of models with rigorous syntax and semantics definitions to:
    eliminate ambiguity and subjective assumptions and
    conduct systematic/mathematical analysis and verification
  - Thus, aims at quality assurance in early phases
    (though we also use “formal verification” for program code)

Simple Example: OCL

- OCL (Object Constraint Language)
  - Formal language to add constraints in UML based on first-order logic



```
{?} AcyclicAncestry
{{OCL} self.parents->closure(parents)-
>excludes(self)}
```

```
{?} NoSelfPartnership
{{OCL} self.partner <> self}
```

```
{?} EachChildHasTwoParents
{{OCL} self.children->forAll(child | child.parents->size() = 2)}
```

```
{?} NameIsAlphabetic
{{OCL} self.name.matches('[a-zA-Z]*')}
```

```
{?} EachChildsParentsArePartners
{{OCL} let selfAndPartner = self->including(self.partner) in
self.children->forAll(child | selfAndPartner->includesAll(child.parents))}
```

Example for class diagram

Cited from
[https://help.eclipse.org/oxygen/index.jsp?topic=%2Forg.eclipse.ocl.doc%2Fhelp%2FOCLExamplesforUML.html]

TOC

- V&V
- Formal Methods
- <u>Underlying Theories</u>
- Example Methods

Theory for Sequential Programs: Overview

- Flowchart verification by Floyd

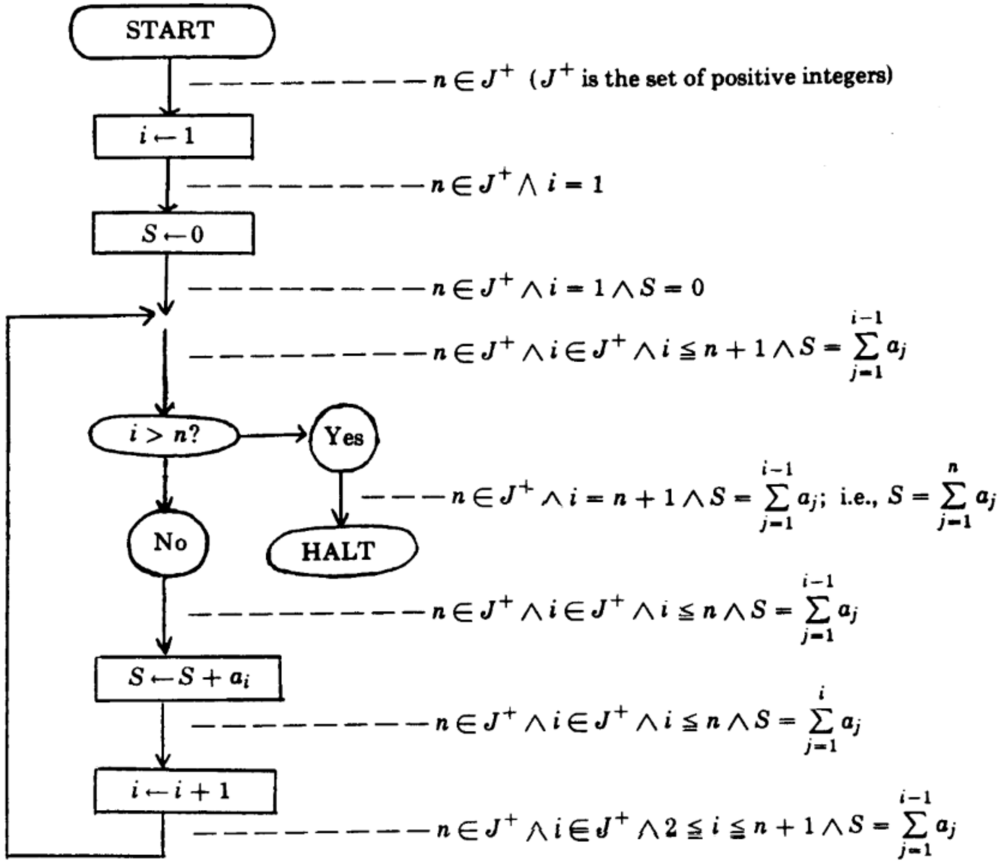


FIGURE 1. Flowchart of program to compute $S = \sum_{j=1}^{n} a_j$ $(n \geqq 0)$

Cited from [Robert W. Floyd, “Assigning Meanings to Programs”]

Theory for Sequential Programs: Hoare Logic (1)

- Hoare Logic
  - Example of axiom

$$\{A[t/x]\}\ x := t\ \{A\}$$

precondition $\{a>0 \land b>0\}$

$x := a$

Postcondition $\{x>0 \land b>0\}$

This triple can be derived from the axiom

  - Example of inference rule

$$\frac{\{C \land A\}\ P\ \{B\} \qquad \{\neg C \land A\}\ Q\ \{B\}}{\{A\}\ \mathtt{if}\ C\ \mathtt{then}\ P\ \mathtt{else}\ Q\ \mathtt{fi}\ \{B\}}$$

Theory for Sequential Programs: Hoare Logic (2)

- Hoare Logic (Cont’d)
  - Another example of inference rule for induction on loops

Assuming the loop invariant A holds at the beginning of one execution of the loop content P, A is preserved after one execution of P

$$\frac{\{C \wedge A\}\ \mathtt{P}\ \{A\}}{\{A\}\ \mathtt{while}\ C\ \mathtt{do}\ \mathtt{P}\ \mathtt{od}\ \{\neg C \wedge A\}}$$

If the above property holds, by induction, we can say the loop invariant A is preserved through the execution of the whole whole statement

Theory for Sequential Programs: Weakest Precondition

- Matching a given triple to existing axioms/inference rules is hard
- It is easier to think to ask “what precondition is necessary to ensure the postcondition after execution of the program”

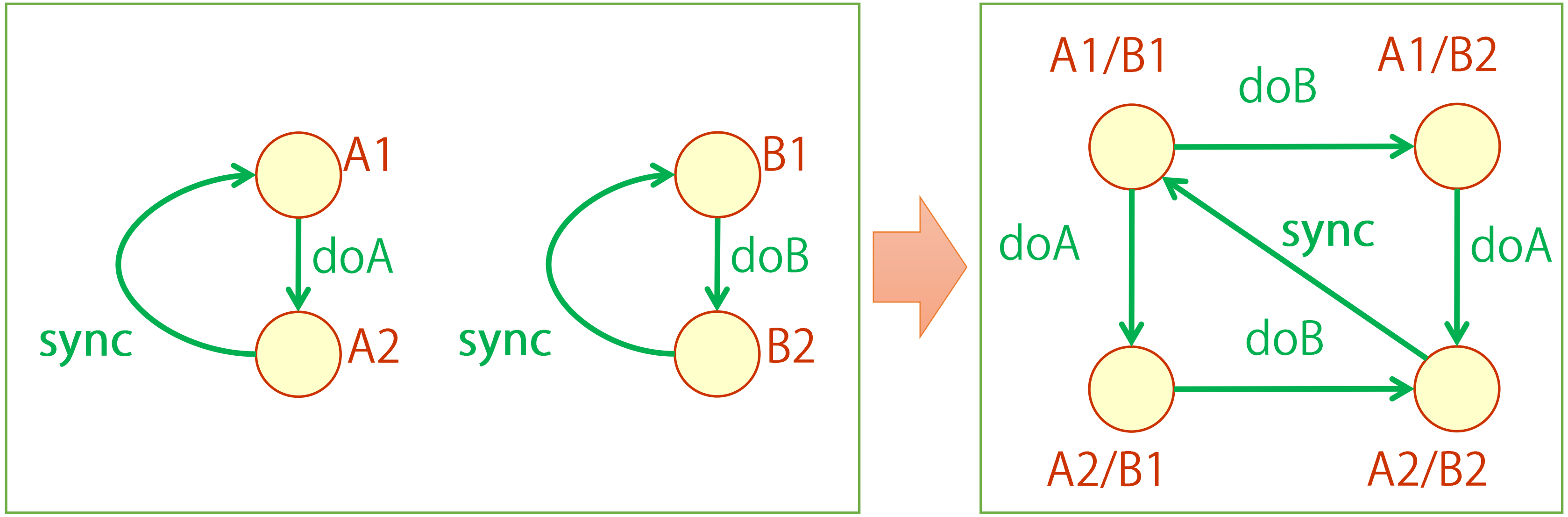
```
wp(x:=t, B) ⇔ B[t/x]

wp(if C then P else Q fi, B) ⇔
     (C⇒wp(P,B))∧(¬C⇒wp(Q,B))
```

wp = weakest precondition

wp = weakest precondition

Theory for Concurrent Systems: Automata

- Consider all the possible states by combining multiple processes



Theory for Concurrent Systems: Temporal Logic

- Specification of temporal properties
- Examples of LTL specifications (Linear Temporal Logic)
  - A and B must not hold at the same time, anytime

    $\square\ !(A \wedge B)$

  - Whenever A holds, B eventually follows

    $\square\ (A \Rightarrow \Diamond B)$

  - A can occur infinitely many times (without infinite blocking)

    $\square \Diamond A$

TOC

- V&V
- Formal Methods
- Underlying Theories
- <u>Example Methods</u>

TOC

- V&V
- Formal Methods
- Underlying Theories
- Example Methods
  - Formal Specification
  - Model Checking
  - Code Verification

f-ishikawa@Sokendai

Formal Specifiction Methods

- Formal Specification Methods（形式仕様記述）
  - Approach oriented to specify and verify a wide range of specification or design, not only specific parts
  - Uses generic formal languages with strong expressiveness including set theories
  - VDM, B-Method, Event-B, Alloy, CafeOBJ, Maude, …

Example of Specification in B-Method (1)

```
MACHINE
    EventManager(capacity)

CONSTRAINTS
    capacity : NAT

SETS
    USERS

VARIABLES
    registered_users

INVARIANT
    registered_users : POW(USERS) &
    card(registered_users) <= capacity
```

Later refined into program-level types, e.g., int arrays

Example of Specification in B-Method (2)

```
INITIALISATION
    registered_users := {}

OPERATIONS
    register(user) =
    PRE user : USERS & user /: registered_users &
        card(registered_users) <= capacity - 1
    THEN registered_users := registered_users ¥/ {user}
    END

END
```

/: means ∉

¥/ means set

Verification in B-Method: Theorem Proving

- Theorem proving based on Hoare Logic
  - The initial state satisfies the invariants
  - The invariants are preserved by all the operations with the valid operation call (the invariants and preconditions satisfied)

➡ By induction, the invariants hold in all possible states

Verification in B-Method: Refinement

- Refinement
  - Models are refined into more concrete ones, i.e., models with more implementation-oriented representations
  - Consistency is checked: the concrete model never reach states that the abstract model does not reach,
    i.e., the invariants of the abstract model are preserved
- Correctness by construction
  - By step-wise refinement, we obtain code and “we already know it is correct”

f-ishikawa@Sokendai

Application Examples

- B-Method is well known in railway systems
  - Automated shuttles in the Paris (CDG) airport
  - Automated metro No. 14 in Paris
  - (then exported to many train systems in the world)
- In Japan, VDM is well-known with FeliCa application
  - VDM is similar to B-Method but more lightweight (program-like syntax, verification by testing)
  - The specification of the IC chips is given with VDM, which is the input to chip vendors

TOC

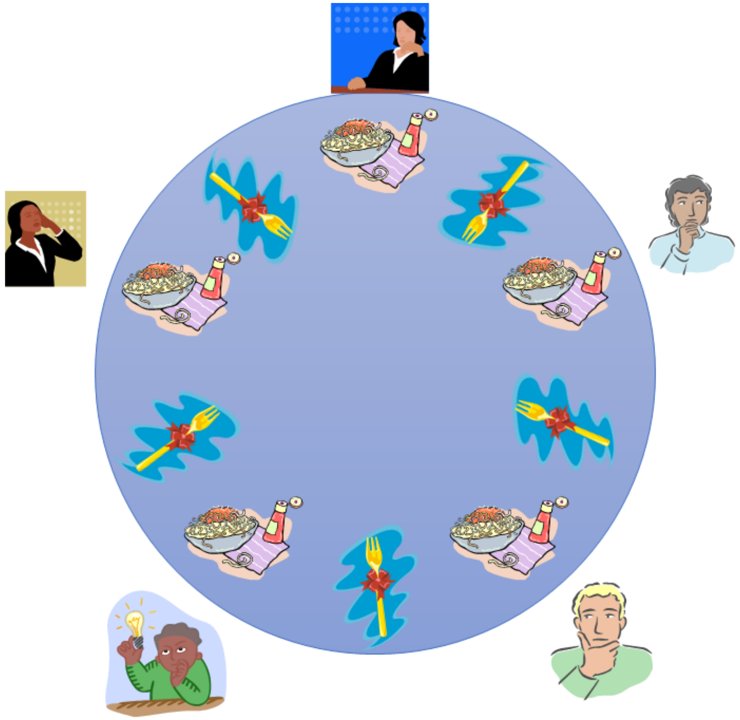
f-ishikawa@Sokendai

Difficulties in Concurrency

- Problems that come to the surface only with very specific execution order/timing

- The dining philosopher problem
  - If each philosopher (process) takes
    - Take the right fork, then the left one, eat, put the left fork, and put the right one

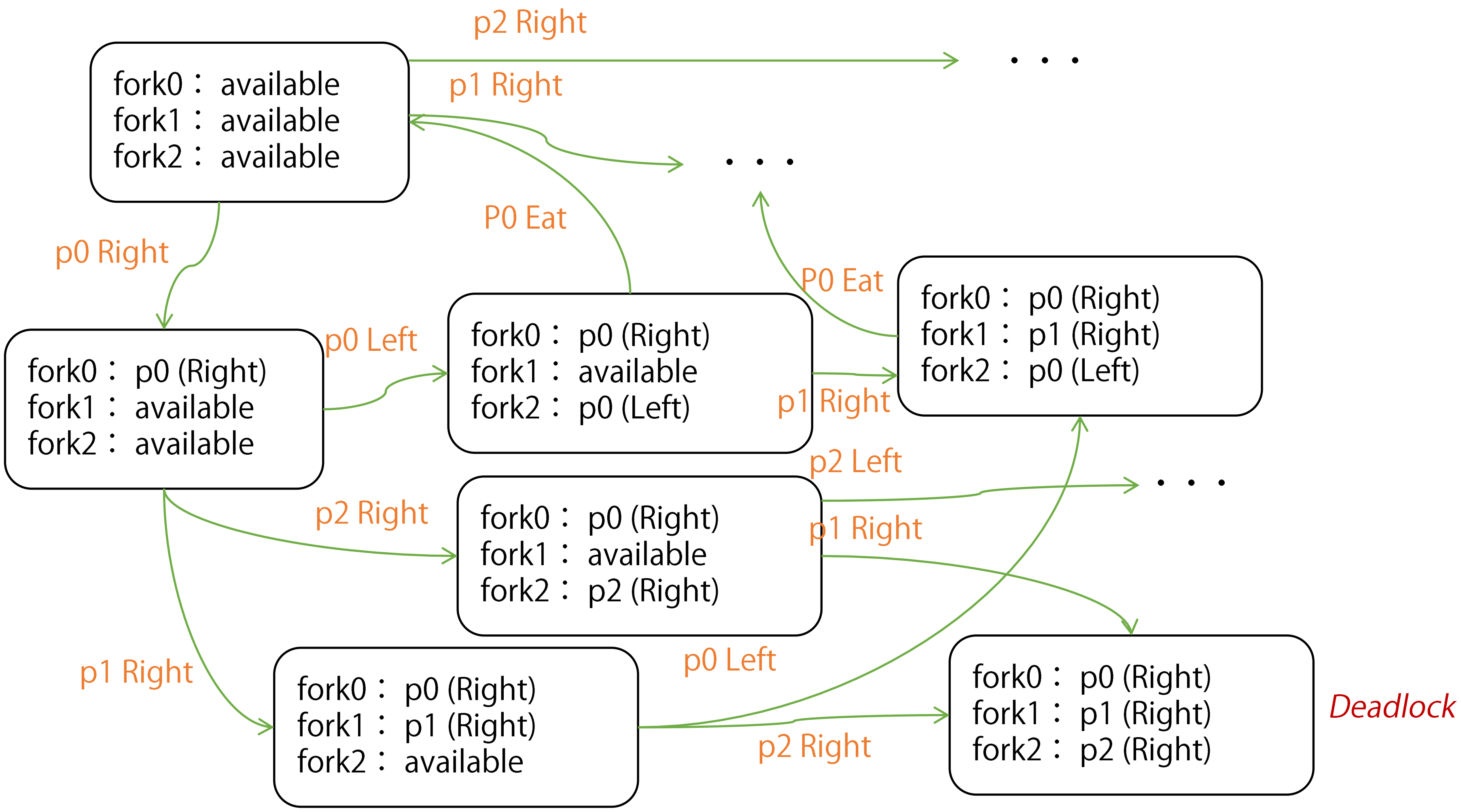
  → Possibility of deadlock



Model Checking

- Model Checking（モデル検査）
  - Intuitively and practically, verification of given properties by exhaustive search over all the possible state transitions
    - Originally, based on a mathematical term “model” refers to an interpretation (e.g., variable assignment) that satisfies a logical formula
  - Useful especially in concurrent systems
  - “One-button” techniques but with the state-explosion problem
  - Need to focus on essences such as control flags, abstracting away unnecessary values in large integers

State Transitions of Dinning Philosophers



p2 Right

・・・

p1 Right

fork0 : available
fork1 : available
fork2 : available

・・・

P0 Eat

p0 Right

P0 Eat

fork0 : p0 (Right)
fork1 : p1 (Right)
fork2 : p0 (Left)

fork0 : p0 (Right)
fork1 : available
fork2 : available

p0 Left

fork0 : p0 (Right)
fork1 : available
fork2 : p0 (Left)

p1 Right

p2 Left

・・・

p2 Right

fork0 : p0 (Right)
fork1 : available
fork2 : p2 (Right)

p1 Right

p1 Right

p0 Left

fork0 : p0 (Right)
fork1 : p1 (Right)
fork2 : available

p2 Right

fork0 : p0 (Right)
fork1 : p1 (Right)
fork2 : p2 (Right)

Deadlock

Example of Process Description in the SPIN Tool

```
mtype = {p0, p1, p2, none};
mtype fork[3] = none;

active proctype P0(){
  do
    :: atomic{fork[0] == none -> fork[0] = p0};
       atomic{fork[2] == none -> fork[2] = p0};
       skip;
       fork[2] = none;
       fork[0] = none;
  od
}
```

Enumerate type

do is infinite loop,
: is for non-deterministic choices
(only one choice in this example)

TOC

- V&V
- Formal Methods
- Underlying Theories
- <u>Example Methods</u>
  - Formal Specification
  - Model Checking
  - <u>Code Verification</u>

Code-Level Verification

- Both of theorem proving and model checking
  - Model checking requires proper bounding, e.g., exhaustive search only within 10000 steps
- What to check
  - Application-specific specifications given in formal languages, if given
  - Application-independent properties such as non-occurrence of null reference, zero division, invalid array index, resource leak, etc.

Specification on Code: Example (1)

- Bank account class in JML (Java Modeling Language)

```
public class BankAccount {

  private /*@ spec_public @*/ int balance;
  private /*@ spec_public @*/ static int MIN_BALANCE = 0;

  //@ public invariant balance >= MIN_BALANCE;

  //@ requires amount > 0;
  //@ requires amount <= balance - MIN_BALANCE;
  //@ ensures balance == ¥old(balance) - amount;
  //@ signals (Exception) amount > balance - MIN_BALANCE;
  public void withdraw(int amount) throws Exception{
    if (balance - amount < MIN_BALANCE) throw new Exception();
    balance = balance - amount;
  }

}
```

Specification on Code: Example (2)

- Binary Search in JML (Java Modeling Language)

```
//@ requires a != null;
//@ requires ¥forall int i; 0 <= i && i < a.length - 1; (¥forall int j; i < j && j< a.length; a[i] < a[j]);
//@ ensures ¥result >= 0 ==> ¥result < a.length && a[¥result] == key;
//@ ensures ¥result < 0 ==> (¥forall int i; 0 <= i && i < a.length; a[i] != key);
public static int binarySearch(int a[], int key) {
    int low = 0;
    int high = a.length;
    //@ maintaining 0 <= low && low <= a.length && 0 <= high && high <= a.length;
    //@ maintaining (¥forall int i; 0 <= i && i < low; a[i] < key);
    //@ maintaining (¥forall int i; high <= i && i < a.length; a[i] > key);
    //@ decreases high - low;
    while (low < high) {
        int mid = low + (high - low) / 2;
        int midVal = a[mid];
        if (key < midVal) { high = mid; }
        else if (midVal < key) { low = mid + 1; }
        else { return mid; // key found}
    }
    return -low - 1; // key not found.
}
```

Specification on Code

- Example of specification language and tool
  - ACSL/Frama-C (for C) [https://frama-c.com/]
  - JML/OpenJML (for Java) [https://www.openjml.org/]
- Typical tool functions
  - Test generation: rewrite the code to include checking of preconditions, postconditions, and invariants
  - Theorem proving based on weakest precondition calculus

Typical Tools for Static Analysis

- Static analysis tools
  (static: without code execution)
  - Often checks only application-independent properties
  - Sometimes theorem proving used inside
    → Possibility of false-negative ("I tried to prove this variable is not null but I cannot find a proof, so I'm making a warning")
- Example: infer (by Facebook)
  - Strong background with Separation Logic (extension of Hoare Logic to handle pointer issues)

[https://fbinfer.com/]

[https://research.fb.com/publications/moving-fast-with-software-verification/]

Summary

- V&V
  - Core activities for quality assurance
  - Distinguishing verification and validation
- Formal Methods
  - Makes use of models with rigorous syntax and semantics definitions
  - Provides strong verification capabilities but also contributes to elimination of unclear or ambiguous descriptions