Formal specifications

a formal specification is the statement of what a system has to do written in a formal language and it is a reference for those who have to develop the system

Problem of the correctness of the formal specification with respect to the informal specification of the system:

$\rightarrow$ first limit (unavoidable) in the application of formal methods

- generally more abstract than the implementation
- a mathematical formalism is used
- complete problem specification, without implementation details

Simple examples of formal specification

1. *Abstract data type: STACK of integer*

   operations: top pop push empty axioms:
   empty(nil) = true
   empty(push(x,S)) = false
   pop(push(x,S) = S
   top(push(x,S)) = x

2. *Square root*

   $\sqrt{x}$ is such that:

   $\sqrt{x}$ * $\sqrt{x}$ = x

Abstract specifications
Axiomatic, algebraic specifications

Simple examples of formal specification

implementation

STACK

array and pointer to the head of the stack

Square root

standard algorithm

Formal specifications

often the specification requires the same effort as the implementation

Example: factorial

formal specification:

$$0! = 1 \quad x! = x * (x - 1)! \quad \textit{for} \quad x > 0$$

Pascal function:

```
function fact(x:integer): integer
 begin
   if x = 0
   then fact := 1
   else fact := x * fact(x-1)
 end
```

The formal specification may be executable (e.g. Prolog)
$\rightarrow$ rapid prototyping

Formal specifications

Formalism/specification language tipology

- *declarative* specifications
  gives a definition of the application in an abstract way, by defining the properties which must be satisfied
  define what a system or component should do, but not how it should do it.

- *operational* specifications
  the application is considered as an abstract machine, based on the concept of state; the machine evolves by changing its state; the specification describes how the machine evolves
  possibly define what a system or component should do, and define at least in part how it should do it.

Formal specifications

The use of formal methods related to the different phases of the software life cycle

- formal specification as a mean to specify in a precise way the system requirements
  $\rightarrow$ in the relation between the client and the customer
  $\rightarrow$ in the comprehension of the problem by the engineers
  $\rightarrow$ as a document in the functional life of the system

- formal specification from which the test cases for the testing phase of the product are derived with the aim of studying its matching with the formal specification.
  $\rightarrow$ Automatic/standard methods to derive test cases

Formal specifications

- executable formal specification: a first prototype of the system from which the final specification must be derived. The final implementation must respect the prototype except for efficiency issues.
  $\rightarrow$ simulation/animation of the formal specification

- from the formal specification a correct implementation is derived in an automatic/standard way
  The formal specification is refined to specifications more related to the implementation, by successive refinement steps.

- Once the implementation has been generated, the correctness of the implementation with respect to the specification is verified.

Formal methods supported by automated tools

Correctness of the implementation with respect to the specification can be verified by means of

- verification algorithms (automatic/semi-automatic)
  (Verification equivalence/ preorder)
  (Model checking)
  (Reachability Analysis)
  es. verification of safety/liveness property expressed by temporal logic formulae on the finite state machine which represents the implementation

- Theorem Proving
  manual
  partially automatic/ automatic es. proof of invariants of a program, by assertions in the Hoare logic

Hoare logic

a formal system with a set of logical rules for reasoning about the correctness of computer programs

Hoare triple: $\{P\}C\{Q\}$
P, Q assertions, S command
Assertion: formulae in predicate logic
Example: $\{x = 1\}\ \ x := x + 1\ \ \{x = 2\}$

When the pre-condition is met, executing the command establishes the post-condition. Hoare provides axioms anf inference rules for all constructs of a simple imperative programming language

Example:
Formal proof
$(x + 1)^2 = x^2 * 2x + 1$
$(x + 1)^2 = (x + 1) * (x + 1)$ by definition of $()^2$
$(x+1)^2 = (x + 1) * x + (x + 1) * 1$ by distributive law of * over +
...

Limits to the applicability of formal methods

Theoretical limits

- limits in the correctness of the passage from the informal specification to the formal one
- limits to the possibility of modeling physical phenomena
- Indecidability of many verification problems in the general case (es. equivalence verification between two general programs) (verification of the correctness of the compiler of a programming language, also in the case in which a formal specification of the language is given)

Limits to the applicability of formal methods

Practical limits

- specification must be generated (readability, comprensibility)
- complexity of the verification algorithms, often not tractable also in the case of reasonable real problems.
- verification tools correctness
- formal semantics of the programming languages (necessary for the verification of the implementation)
- Implementation details implications:
  e.g. $\sqrt{2} * \sqrt{2} \neq 2$
  e.g. the array implementation of the stack does not respect the axioms when the stack is full

Declarative formalisms

- start from an abstract description of the state space and they are not related to machines or automata
- describe the reachable states of the system in a general and implicit way
- use mathematical concepts (equations, axioms, constraints and properties) expressed by logical or algebraic formalisms
- the next state relation is not specified

Logical formalisms:

- *Propositional logic* (atomic propositions, relation between propositions - $\wedge, \vee, \neg, \cdots$)
- *First order logic* (quantified variables - predicates and quantifier $\forall$, $\exists$)
- *Higher-order logic* (predicates quantifier and function quantifier)
- Temporal logic - (propositions qualified in terms of time (always, eventually, $\cdots$)

Declarative formalisms

Axiomatic/ Algebraic formalisms:

- Z, B, VDM
  $\rightarrow$ pre-post conditions
  (conditions which are true before and after the execution of an operation)
  $\rightarrow$ invariants
  (conditions which are always true).
  Operations are the main concern.

- ACT ONE, OBJ
  $\rightarrow$ algebraic specification of Abstract Data Types (see the Stack).
  Data types are the main concern.

Declarative formalisms

Functional formalisms:

- Isabelle, PVS
  $\rightarrow$ state the desired computation defining a function that will achieve the computation. The computation is stated in a declarative way.
  Higher-order logic proof tools.

Operational formalisms

- the system specification is done by giving the states reachable by the system during its evolution and by giving the transitions which lead from a state to another state

Operational formalisms:

- *Finite state automata*
- *Petri nets*
- *Process algebras*

An example of declarative specification: sorting an array

A declarative specification will state that the program terminates and that, after its termination, the array is well sorted.

Declarative specification in the formal language Z, proposed by Abrial in 1977.

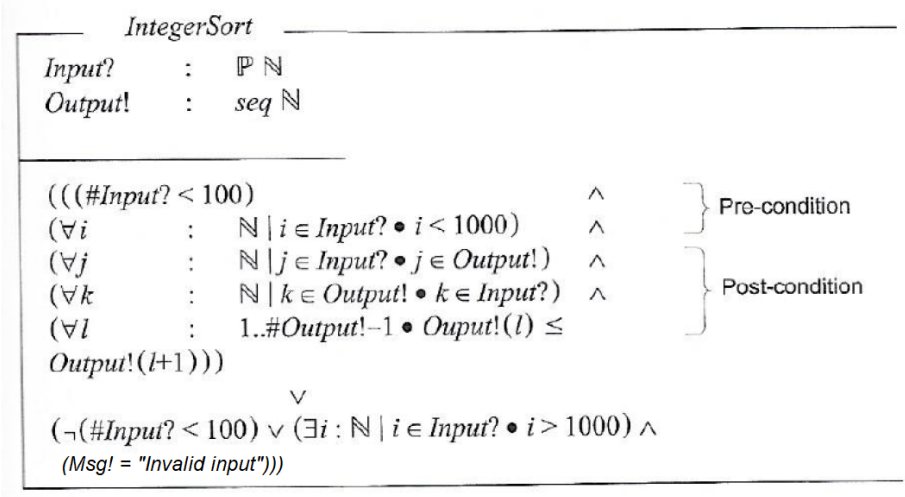
Z, primarily developed at the university of Oxford, used widely in industrial applications.

Notation is identical to traditional discrete mathematics

John Knights.
Fundamentals of Dependable Computing.Chapman & Hall, 2012 (chapt.8)

An example: sorting an array

Let us consider an array of integer numbers.
Specification written in Z.



An example: sorting an array

using discrete mathematics in a formal specification is not especially difficult

Inferring the meaning of the specification:

- the name at the top specifies what the specification is about (integer sorting)
- the lables "Pre-condition" and "Post-condition" are not part of the specification
  Pre-condition identifies the condition that must hold before the specification can be applied
  Post-condition identifies the condition that must hold after the specification has been applied.
- the cardinality of the set Input? has to be less than 100; i.e., the specification is stating that the associated program has to deal with less than 100 numbers to sort.
- ...

Analysis of properties

- Analysis through speculative theorems can be conducted to determine whether the specification satisfies certain expected properties.
  For example, certain states are never reached, certain variables maintain a certain relationship, certain events can never occurr, $\cdots$
- The specification is taken as a set of axioms.
  The property is stated as a theorem. The theorem can be proved from the specification.
- if the implementation implements correctly the specification, then the implementation has tha same property.

The clarity of the formal language allows effective human review of the specification.

The language

- The main structuring mehanism is the schema. Then we have space: vertical, between lines, between operators and operand, tab lines with overall the same layout (e.g. declarations) , $\cdots$
- use the structuring mechanisms
- use meaningful identifiers to help explain the meaning of the specification
- the language makes use of propositional calculus, predicate calculus, quantifiers (for all, exists), set theory, sequences, functions, relations, $\cdots$

The language

- the equal sign has the meaning of equality
  a = b+c
  is a predicate that return true or false depending on the value of a, b, and c.
  declarative languages do not include the statement of assignment

- reference to pieces of the state before the execution of an operation, and after the execution of the operation (this allows to model the effect of an operation), typically the prime symbol is used (v and v')
  For example: $v' = v + 1$
  means that the after state of v is equal to $v + 1$

Square root

Natural language: *the program shall read one positive input and write one output that is the integer square root of the input.*

Declarative specification:

$$(x > 0) \wedge (x >= (y')^2) \wedge (x < (y' + 1)^2)$$

- property of the input in order to apply the operation
- what we want the final state to be

$x = 5$
$y' = 2$
$(5 >= (2)^2)$ and $(5 < (3)^2)$
$(5 >= 4)$ and $(5 < 9)$

$y' = 3$
$(5 >= (3)^2)$ and $(5 < (4)^2)$
$(5 >= 9)$ and $(5 < 16)$

Square root

Procedural specification:

```
integer x;
read (x);
if (x>0) then
    y:=x;
    while y^2 > x loop
      y:=y-1;
    end loop;
end if ;

print (y);
```

- how the computation should be carried out

The schema

The schema is a means to organise specifications.

A schema in divided in two parts, which can access to both before-state and after-state variables:

- the *signature*
  where variables are declared
- the *predicate*
  a predicate on the variables
  - schema whose predicates do not include after-state variables, define an element of the state
  - schema whose predicates include after-state variables, are used to define operations
  - the predicate in a state defines an invariant on the state. Since the invariant may include several conjunctions each involving a different variable, the predicate in a state schema is commonly thought of as a collection of invariants.

The schema calculus

A complex calculus that includes:

- schema composition
- schema expressions
- schema types

An example of decoration of schemas is Δ

ΔCounter is a new schema equal to the schema Counter, except that all variables are primed. The invariant is included with also variables primed.

The schema calculus

Counter

$$CounterValue : \mathbb{N}_1$$

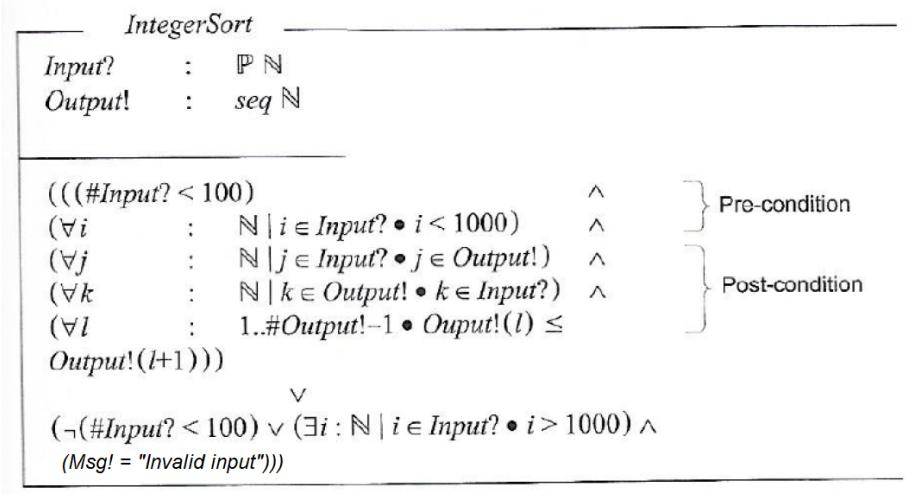
$$CounterValue \leq 10$$

ΔCounter

$$CounterValue, CounterValue' : \mathbb{N}_1$$

$$CounterValue \leq 10$$
$$CounterValue' \leq 10$$

Integer Sort example



Integer Sort example

The specification:

- single schema
- signature: input variable Input?, a set of natural numbers and output variable Output!, a sequence of natural numbers
- predicate: lines 1, 2 and 7 are pre-conditions for the operation; the others are post-conditions (except line 6, an operator).
- line 5 states that: all except the last number in the sequence, must be less than or equal to the next number in sequence
- line 8 states that: if the pre-condition is false a message must be printed.

Note: the input is stated to be a set, therefore we assume that we do not have replicated values.