IX. System Models (III)

Objective

- Introduce formal descriptive notation
  - Logic
  - Algebraic
  - Z

Logic Specifications

- Use of **First Order Logic** formula to describe program properties
  - **Expressions involve variables, numeric constants, functions, predicates. Logical connectives and, or, not, implies, for all, exists**
  - Examples

    x > y **and y > z implies** x>z

    for all i **in** N x[i] > x[i+1]

    for all M **in** N (**exist** n **in** N(n>M))
- Free variables and bound variables

Logic Specifications – input/output assertions

- A property for a program or a subprogram P is specified as a formula of type

  $\{Pre(i_1,i_2,\ldots,i_n)\}$

  P

  $\{Post(O_1,O_2,\ldots,O_m,i_1,i_2,\ldots,i_n)\}$

input/output assertions - example

- {true}

  P

  $\{(o=i_1$ **or** $o=i_2)$ **and** $o \geq i_1$ **and** $o \geq i_2\}$
- {n>0}

  P

  $\{o=\sum_{k=1\ldots n} i_k\}$
- $\{(i_1>0$ **and** $i_2>0)\}$

  P

  $\{($**exists** $\mathbf{z}_1, z_2\ (i_1=o \times \mathbf{z}_1$ **and** $i_2=o \times z_2)$ **and not**

  $($**exists** $h(i_1=h \times \mathbf{z}_1$ **and** $i_2=h \times z_2)$ and $h>o))\}$

Generalising input/output assertions

- We would like to talk of **programs fragments**!
- The language should be able to **refer variables defined within the program**

- {n > 0} – n is a constant value

  procedure search(table: **in** integer_array; n: **in** integer;
  element: **in** integer; found: **out** Boolean);

  {found**=(exist** i(1<=i<=n and table(i)=element))**}**

input/output assertions and classes

- The language "predicates" over object variables and data structures defining:
- **Invariants**
  - Object states
  - Methods execution must preserve invariants
    - Examples: set abstraction or ordered set abstraction
- **Pre- and Post-conditions on each methods**
  - {INV **and** precondition} program fragment for m {INV **and** post-condition}

- Logical statements **defined by the developer** to increase verification power

Example

```
public class Queue {
  Object[] queue;
  int length,head,tail,elementInQueue;
  public Queue(int length) {
    queue = new Object[length];
    this.length=length;this.head=0;this.tail=0;this.elementInQueue=0;
  }
  public void insert(Object o) {
    queue[head]=o;head++;elementInQueue++;
    if (head==length) head=0;
  }
  public Object remove() {
    Object o = queue[tail];tail++;elementInQueue--;
    if (tail==length) tail=0;
    return o;
  }
  public boolean isEmpty() {
    return (head==tail);
  }
}
```

Design by Contract

- Engineering principle known as Design by Contract (DbC)
  - Logical statements defined at the interface level
  - Constitute the agreement among the contractor and the client
  - Support in programming languages: Eiffel, JContractor ...

- Design vs. Requirements using logic specifications
  - Which is the universe of discourse?

Verification of Specifications

- Using FOT you specify that **any implementation must guarantee** that all of the given rules are true
- Logical expressions can be used to analyze system properties **applying logical deduction**
  - **Derive the formula from the specification of the system**
- Operational formalisms do not allow to prove properties
- You can discover behaviour but you cannot provide a proof!

- Proving theorems is not a decidable problem within FOT

Algebraic Specification

- Large systems are decomposed into subsystems with well-defined interfaces between these subsystems.
- Specification of **subsystem interfaces allows independent development of the different subsystems**.
- Interfaces may be defined as abstract data types or object classes.
- The **algebraic approach to formal specification is particularly well-suited to interface specification as it is focused on the defined operations in an object**.

Specification elements

- Introduction
  - Defines the sort (the type name) and declares other specifications that are used.
- Description
  - Informally describes the operations on the type.
- Signature
  - Defines the syntax of the operations in the interface and their parameters.
- Axioms
  - Defines the operation semantics by defining axioms which characterise behaviour.

Systematic Algebraic Specification

- Algebraic specifications of a system may be developed in a systematic way
  - Specification structuring;
  - Specification naming;
  - Operation selection;
  - Informal operation specification;
  - Syntax definition;
  - Axiom definition.

Specification Operations

- **Constructor operations**. Operations which create entities of the type being specified.
- **Inspection operations**. Operations which evaluate entities of the type being specified.
- Rule of thumb: identify constructor operations and define inspector operations for each primitive constructor operation.

Operations on a List ADT

- Constructor operations which evaluate to sort List
  - Create, Cons and Tail.
- Inspection operations which take sort list as a parameter and return some other sort
  - Head and Length.
- Tail can be defined using the simpler constructors Create and Cons. No need to define Head and Length with Tail.

Operations on a List ADT

LIST(ELEM)

sort List
imports INTEGER

Defines a list where elements are added at the end and removed from the front. The operations are Create, which brings an empty list into existence, Cons, which creates a new list with an added member, Length, which evaluates the list size, Head, which evaluates the front element of the list, and Tail, which creates a list by removing the head from its input list. Undefined represents an undefined value of type Elem.

Create --> List
Cons(List, Elem) --> List
Head(List) --> Elem
Length(List) --> Integer
Tail(List) --> List

Head(Create) = Undefined **exception** (empty list)
Head(Cons(L,v)) = **if** L = Create **then** v **else** Head(L)
Length(Create)=0
Length(Cons(L,v)) = Length(L) + 1
Tail(Create) = Create
Tail(Cons(L,v)) = **if** L = Create **then** Create **else** Cons(Tail(L),v)

Operations on a List ADT

- Operations are often specified recursively.
- Tail (Cons (L, v)) = **if** L = Create **then** Create

  else Cons (Tail (L), v).
  - Cons ([5, 7], 9) = [5, 7, 9]
  - Tail ([5, 7, 9]) = Tail (Cons ([5, 7], 9)) =
  - Cons (Tail ([5, 7]), 9) = Cons (Tail (Cons ([5], 7)), 9) =
  - Cons (Cons (Tail ([5]), 7), 9) =
  - Cons (Cons (Tail (Cons ([], 5)), 7), 9) =
  - Cons (Cons ([Create], 7), 9) = Cons ([7], 9) = [7, 9]

Algebraic specifications

- Algebraic Specifications can be easily combined
- **Reuse of algebras**
  - **Import allow to reuse**
  - **Assume allows to rewrite definition**

Keynote

- Discussed two formalism for defining a software specification with a descriptive flavour
  - Use of FOT
  - Algebraic specification
- Operational spec better for simulation descriptive for analysis
- Limitations of analysis have been highlighted