Descriptive specifications

Topics:
- Logic specifications

Logic specifications

Examples of first-order theory (FOT) formulas:
- \( x > y \land y > z \Rightarrow x > z \)
- \( x = y \iff y = x \)
- \( \forall x, y, z : (x > y \land y > z \Rightarrow x > z) \)
- \( x + 1 < x - 1 \)
- \( \forall x : (3y : y = x + z) \)
- \( x > 3 \lor x < -6 \)

Specifying complete programs

A property, or requirement, for \( P \) is specified as a formula of the type

\[
\{ \text{Pre} \ (i_1, i_2, \ldots, i_n) \} \quad P \quad \{ \text{Post} \ (o_1, o_2, \ldots, o_m, i_1, i_2, \ldots, i_n) \}
\]

Pre: precondition
Post: postcondition

Example

Program \( P \) that computes greatest common divisor of two integers \( i_1 \) and \( i_2 \)

\[
\{ i_1 > 0 \land i_2 > 0 \} \quad P \quad \{ \exists z_1, z_2 : i_1 = o * z_1 \land i_2 = o * z_2 \} \\
\land \neg \exists h > o : (\exists z_1, z_2 : i_1 = h * z_1 \land i_2 = h * z_2) \}
\]

Specifying procedures

\[
\{ n > 0 \} \quad \text{procedure} \ \text{search} \ (\text{table}: \text{integer}_\text{array}; \ n: \text{integer}; \ \text{element}: \text{integer}; \ \text{found}: \text{out} \ \text{bool}); \quad \{ \text{found} = (\exists i : 1 \leq i \leq n \land \text{table}(i) = \text{element}) \}
\]

\[
\{ n > 0 \land 8a = n \} \quad \text{procedure} \ \text{reverse} \ (a: \text{in} \ \text{out} \ \text{integer}_\text{array}; \ n: \text{in} \ \text{integer}); \quad \{ \forall i \in (1..n) : a(i) = \text{old}-a(n - i + 1) \}
\]

Invariant

Predicate stating a condition that must always hold
- Many forms:
  - Data invariants
  - Path invariants
- Powerful form of documentation

Example: data invariant stating that an array \( \text{IMPL} \) implements the ADT set

\[
\forall i, j : (1 \leq i \leq \#\text{IMPL} \land 1 \leq j \leq \#\text{IMPL} \land i \neq j) \\
\Rightarrow \text{IMPL}[i] \neq \text{IMPL}[j]
\]

Says that no duplicates are stored
Specifying non-terminating behaviors

Example: producer+consumer+buffer

Invariant specifies that whatever has been produced is the concatenation of what has been taken from the buffer and what is kept in the buffer

\[ \text{input\_sequence} = \text{append} (\text{output\_sequence}, \text{contents(CHAR\_BUFFER)}) \]

Case study: Elevator example

Elementary predicates

- **at** \((E, F, T)\)
  - \(E\) is at floor \(F\) at time \(T\)
- **start** \((E, F, T, \text{up})\)
  - \(E\) left floor \(F\) at time \(T\) moving up

Rules

- \((\text{at} (E, F, T) \land \text{on} (ER, F_1, T) \land F_1 > F )\) \Rightarrow \text{start} (E, F, T, \text{up})
  - When \(E\) is at floor \(F\) at time \(T\) and the elevator button listing floor \(F_1\) is on and \(F_1 > F\), then \(E\) starts moving up

States and events

Elementary predicates are partitioned into

- states, having non-null duration
  - \(\text{standing}(E, F, T_1, T_2)\)
    - assumption: closed at left, open at right
- events
  - instantaneous (caused state change occurs at same time)
    - represented by predicates that hold only at a particular time instant
    - **arrival** \((E, F, T)\)
      - \(E\) is at floor \(F\) at time \(T\)

For simplicity, we assume

- zero decision time
- no simultaneous events

Events (1)

**arrival** \((E, F, T)\)

- represents arrival of elevator \(E\) at floor \(F\) at time \(T\)
  - does not say if it will stop or will proceed, nor where it comes from
  - \(E \in \{1..n\}, F \in \{1..m\}, T \geq t_0\) (\(t_0\) is the initial time)

**departure** \((E, F, D, T)\)

- represents departure of \(E\) from floor \(F\) in direction \(D\) at time \(T\)
  - \(E \in \{1..n\}, F \in \{1..m\}, D \in \{\text{up, down}\}, T \geq t_0\)

**stop** \((E, F, T)\)

- \(E \in \{1..n\}, F \in \{1..m\}, T \geq t_0\)
  - specifies stop to serve an internal or external request

Events (2)

**new** \(\text{list} (E, L, T)\)

- List of floors for \(E\) becomes \(L\) at time \(T\)
  - \(L \in \{1..m\}^*\)
  - \(E \in \{1..n\}\)
  - \(T \geq t_0\)
  - \(L\) is the list of floors to visit
  - scheduling performed by system control component

**call** \((F, D, T)\)

- external call
- summons some elevator \(F\) in direction \(D\)

**request** \((E, F, T)\)

- internal reservation
- user within elevator \(E\) presses button for floor \(F\)

States

**moving** \((E, F, D, T_1, T_2)\)

- \(E\) is moving from floor \(F\) in direction \(D\) during the interval \([T_1, T_2)\)

**standing** \((E, F, T_1, T_2)\)

**list** \((E, L, T_1, T_2)\)

- \(L\) is the list of floors for \(E\) in \([T_1, T_2)\)

We implicitly assume that state predicates hold for any sub-interval (i.e., the rules that describe this are assumed to be automatically added)

- Nothing prevents that it holds for larger interval
Rules relating events and states

R1: When an elevator (E) arrives at floor F, it continues to move if there is no request for service from F and the list is not empty. Furthermore, if the floor to serve is higher, it moves upward.

\[
\text{arrival}(E, F, T_a) \land \text{list}(E, L, T, T_a) \land \text{first}(L) > F \\
\implies \text{departure}(E, F, \text{up}, T_a)
\]

A similar rule (R1.1) describes downward movement.

R2: Upon arrival at F, if F must be serviced (F appears as first of the list), E stops.

\[
\text{arrival}(E, F, T_a) \land \text{list}(E, L, T, T_a) \land \text{first}(L) = F \\
\implies \text{stop}(E, F, T_a)
\]

R3: E stops at F if it gets there with an empty list.

\[
\text{arrival}(E, F, T_a) \land \text{list}(E, \text{empty}, T, T_a) \\
\implies \text{stop}(E, F, T_a)
\]

Elevators have a fixed time (\(D_{ts}\)) to service a floor. If list is not empty at the end of such interval, the elevator must leave the floor immediately.

\[
\text{stop}(E, F, T_a) \land \text{list}(E, L, T, T_a + D_{ts}) \land \text{first}(L) > F \\
\implies \text{departure}(E, F, \text{up}, T_a + D_{ts})
\]

Rule R4

R4: If the elevator has no floors to service, it should stop until its list becomes nonempty.

\[
\text{stop}(E, F, T_a) \land \text{list}(E, \text{empty}, T, T_a + D_{ts}) \\
\land \text{list}(E, L, T, T_p) \land T_p > T_a + D_{ts} \land \text{first}(L) > F \\
\implies \text{departure}(E, F, \text{up}, T_p)
\]

Rule R5

R6: Assume that the time to move from one floor to the next is known and fixed. The rule describes movement.

\[
\text{departure}(E, F, \text{up}, T) \\
\implies \text{arrival}(E, F + 1, T + D_t)
\]

R7: The event of stopping initiates standing for at least \(D_{ts}\).

\[
\text{stop}(E, F, T) \\
\implies \text{standing}(E, F, T, T + D_{ts})
\]

R8: At the end of the minimum stop interval, \(D_{ts}\), E remains standing if there are no floors to service.

\[
\text{stop}(E, F, T) \land \text{list}(E, \text{empty}, T, T + D_{ts}) \\
\implies \text{standing}(E, F, T, T + D_{ts})
\]

R9: Departure causes moving.

\[
\text{departure}(E, F, D, T) \\
\implies \text{moving}(E, F, D, T, T + D_t)
\]
**Control rules**

Express the scheduling strategy (by describing “new_list” events and “list” states)

Internal requests are inserted in the list from current floor to top if the elevator is moving up.

External calls are inserted in the list of the closest elevator that is moving in the correct direction, or in a standing elevator.

---

**Note:**
Adapted from Ghezzi, Jazayeri, and Mandrioli

### Control rules

**R10:** Reserving $F$ from inside $E$, which is not standing at $F$, causes immediate update of $L$ according to previous policy.

\[
\text{request}(E, F, T_e) \land \\
\neg \text{standing}(E, F, T_e, T_d) \land \\
\text{list}(E, L, T, T_d) \\
\Rightarrow \\
\text{new_list}(E, L, T, T_d)
\]

---

**R11:** Effect of arrival of $E$ at floor $F$.

\[
\text{arrival}(E, F, T_a) \land \\
\text{list}(E, L, T, T_a) \land \\
F = \text{first}(L) \\
\Rightarrow \\
\text{new_list}(E, \text{tail}(L), T_a)
\]

---

**Verifying specifications**

The system can be simulated by providing a state (set of facts) and using rules to make deductions:

- standing $(2, 3, 5, 7)$ (i.e., elevator 2 is at floor 3 from instant $[5, 7)$)
- list(2, empty, 5, 7)
- request(2, 8, 7)
- new_list(2, [8], 7)

\[
\Rightarrow \quad \text{(excluding other events)}
\]

- departure(2, up, $7 + D_t$)
- arrival(2, 8, $7 + D_t + D_a * (8-3)$)

---

**Verifying specifications**

Properties can be stated and proved via deductions:

\[
\text{new_list}(E, L, T) \land F \in L \\
\Rightarrow \\
\text{new_list}(E, L, T) \land F \notin L, \land T > T
\]

(all requests are served eventually)

---

**Descriptive specs**

The system and its properties are described in the same language.

Proving properties, however, cannot be fully mechanized for most languages.