Verificación formal

Facultad de Informática

September 6, 2024

Indice

1. Validación
   - Formas de validar
2. Diseño por contrato
   - Correctitud
   - Especificación
3. Aserciones
   - Tipos de aserciones
   - Sintaxis Java
4. Verificación formal
   - Ternas de Hoare
   - Estados, asertos y expresiones

Tabla de contenidos

1. Validación
   Formas de validar
2. Diseño por contrato
   Correctitud
   Especificación
3. Aserciones
   Tipos de aserciones
   Sintaxis Java
4. Verificación formal
   Ternas de Hoare
   Estados, asertos y expresiones

Validación
Formas de validación

Cómo se puede validar?

1. Dinámica

Nos enfocaremos en la estática

Validación
Formas de validación

Cómo se puede validar?

1. Dinámica
   - Ejecuciones de prueba a partir de conjuntos de datos

Nos enfocaremos en la estática

Validación
Formas de validación

Cómo se puede validar?

1. Dinámica
   - Ejecuciones de prueba a partir de conjuntos de datos
   - Los errores aparecen más tarde

Nos enfocaremos en la estática

Validación
Formas de validación

Cómo se puede validar?

1. Dinámica
   - Ejecuciones de prueba a partir de conjuntos de datos
   - Los errores aparecen más tarde
   - Difícil de determinar dónde aparecen en el programa

Nos enfocaremos en la estática

Validación
Formas de validación

Cómo se puede validar?

1. Dinámica
   - Ejecuciones de prueba a partir de conjuntos de datos
   - Los errores aparecen más tarde
   - Difícil de determinar dónde aparecen en el programa
2. Estática

Nos enfocaremos en la estática

Validación
Formas de validación

Cómo se puede validar?

1. Dinámica
   - Ejecuciones de prueba a partir de conjuntos de datos
   - Los errores aparecen más tarde
   - Difícil de determinar dónde aparecen en el programa
2. Estática
   - Obtención de información a priori en el texto del programa,**(VERIFICACION)**.

Nos enfocaremos en la estática

Validación
Formas de validación

Cómo se puede validar?

1. Dinámica
   - Ejecuciones de prueba a partir de conjuntos de datos
   - Los errores aparecen más tarde
   - Difícil de determinar dónde aparecen en el programa
2. Estática
   - Obtención de información a priori en el texto del programa,**(VERIFICACION)**.
   - Se puede pensar en construir de programas ya verificados desde su inicio **(DERIVACION).**

Nos enfocaremos en la estática

Validación
Formas de validaciòn

Un software es correcto cuando es consistente con su especificación !!

validación...

- Para asegurar que un programa/algoritmo sea correcto se realiza la validación estática o dinámica.

Validación
Formas de validaciòn

Un software es correcto cuando es consistente con su especificación !!

validación...

- Para asegurar que un programa/algoritmo sea correcto se realiza la validación estática o dinámica.
- Testing: Seleccionar un conjunto de valores de entrada para determinar si los resultados coinciden con los valores esperados

Validación
Formas de validaciòn

Un software es correcto cuando es consistente con su especificación !!

validación...

- Para asegurar que un programa/algoritmo sea correcto se realiza la validación estática o dinámica.
- Testing: Seleccionar un conjunto de valores de entrada para determinar si los resultados coinciden con los valores esperados
- Verificar el programa consiste entonces en "demostrar" que cumple su especificación.

Verificación formal
Definiciones

Definición

La verificación formal de programas consiste en un conjunto de técnicas de comprobación formales que permiten demostrar si un programa funciona correctamente.

- **Programa que funciona correctamente:** cumple con unas especificaciones dadas.
- **Técnicas de comprobación:** La verificación consiste en un proceso de inferencia. Por tanto cada tipo de sentencia ejecutable posee una regla de inferencia.
- **Representación formal:** Ternas de Hoare

Tabla de contenidos

1. Validación
   - Formas de validar
2. Diseño por contrato
   - Correctitud
   - Especificación
3. Aserciones
   - Tipos de aserciones
   - Sintaxis Java
4. Verificación formal
   - Ternas de Hoare
   - Estados, asertos y expresiones

Diseño por contrato

En general

- Ve las relaciones entre una clase y sus clientes como un **acuerdo formal** donde se expresan las obligaciones y derechos de cada uno de los participantes.
- Se busca obtener confiabilidad a partir del cumplimiento de las obligaciones y responsabilidades definiendo un software **correcto**.
- Las aserciones forman parte del **diseño por contrato**.

Especificación formal
En módulos

- Código de un módulo.
- Indicamos qué variables representan datos y cuáles resultados.
- La especificación constará de cabecera, pre y postcondición.

```
// cabecera de un módulo //
MODULO nom (parámetros) RETORNA resul
   {Pre}
   variables locales
   {Pos}
   RETORNA resul
FIN MODULO
```

Representación formal
Conceptos

- Por código o parte de un código entendemos desde una sentencia hasta un programa.
- La verificación la realizaremos sobre programas escritos tipo Java:
  - Todos los operadores aritméticos y todas las funciones.
  - Bloques { }
  - Construcciones alternativas *if* y *if* − *then* − *else*
  - Construcciones repetitiva while.
- No se utilizarán declaraciones de tipo
- No se utilizarán sentencias de entrada/salida.
- La concatenación de dos códigos C_1 y C_2 supone la ejecución secuencial de dichos códigos y vendrá representada de la siguiente forma: $C_1; C_2$.

Especificación de módulos

Pre y pos condiciones

Seudocódigo, según el lenguaje podrá retornar uno o varios valores

```
MÓDULO dividir (a, b: entero) RETORNA c
    {Pre : a ≥ 0, b > 0}
    var ... (declaración de variables locales)
    (instrucciones)
    {Pos : a = b * c + r ∧ r < b}
    RETORNA c
FIN MÓDULO
```

Dados dos números naturales a y b, $b \neq 0$, encontrar el cociente entre a y b.

$$P : \{a, b \in \mathbb{N} \land b > 0\}$$
$$Q : \{a = b*c+r \land 0 \leq r < b \land c \geq 0\}$$

Tabla de contenidos

1. Validación
   - Formas de validar
2. Diseño por contrato
   - Correctitud
   - Especificación
3. Aserciones
   - Tipos de aserciones
   - Sintaxis Java
4. Verificación formal
   - Ternas de Hoare
   - Estados, asertos y expresiones

Aserciones
Java

- Las precondiciones y poscondiciones son denominadas también aserciones.
- Las aserciones son expresiones que establecen algunas propiedades que las entidades del software deben cumplir en algún momento de la ejecución.
  - Java provee la sentencia *assert* para definir aserciones.
  - Si bien no soporta el modelo por contrato ayuda a construir una estrategia informal a dicho modelo.

Aserciones
Uso de aserciones

Los lenguajes de programación la utilizan de distinta manera:

- En Eiffel, las aserciones son parte del proceso de diseño
- En Java, son utilizadas para comprobar código en tiempo de ejecución.
- Utilizar aserciones genera un código confiable
- Las aserciones no permiten al programa recuperarse de los errores
- Un fallo de aserción supone una terminación abrupta del programa

Pre y pos condiciones
Cuando se aplican?

- La precondición establece las propiedades a cumplir cada vez que se invoque a la operación.
- La precondición se debe aplicar en todos los mensajes donde aparece la operación.
- La precondición involucra propiedades del objeto receptor y/o de los parámetros.
- La poscondición debe expresar propiedades resultantes de la ejecución de un servicio, siempre que la ejecución funcione correctamente

Aserciones
Sintaxis Java

Sintaxis

```
assert expresion1;
assert expresion1: expresion2;
```

- Expresion1 debe ser una expresión booleana.
- Expresion2 debe producir un texto con un mensaje de error apropiado. Puede ser la invocación de un método pero no de tipo *void*

- Expresion1 al ser evaluada por el assert puede pasar:
  - Si se cumple → sigue la ejecución.
  - Si no se cumple → EXCEPCION de tipo AssertionError.
  - Si no se cumple y hay expresion2 → el resultado se pasa como argumento al constructor del AssertionError lanzado

Aserciones
Compilación Java

- Para compilar una clase con aserciones debe usarse el argumento javac–source 1.4 MiClase
- El control de aserciones puede habilitarse o deshabilitarse. Por defecto están deshabitadas por lo que debe usarse la opción -ea. java –ea MiClase

Ejemplo

```
assert ref != null;
assert saldo == (oldSaldo + cantidad);
assert ref.m1(parametro);
assert valor>0 : “argumento negativo”;
```

Ejemplo de aserciones: Uso de aserciones para las pre y pos condiciones Help

```
... if ( valor == 1){...}
    else if (valor == 2){    }
    else {assert valor == 3    }

public Elemento elegirSiguiente (){
    assert !coleccionVacia ();....
    return(    .);
}

public void agregar (E:elemento){
...assert(!coleccionVacia());

private boolean invariante(){//metodo logico
    return ( (cant >= 0 ) && (cant <capacidad) );
```

Aserciones
Cuando usarlas

No usar aserciones cuando....

- a la entrada de métodos públicos
- para detectar errores en los datos de entrada al programa

Usar aserciones cuando....

- a la entrada de métodos privados
- a la salida de métodos públicos o privados
- para chequear variables y estructuras de datos interna
- en los bucles iterativos para comprobar condiciones de corte
- en la clausula else de construcciones switch
- en las cláusulas alternativas, casos que no deberían ejecutarse

Tabla de contenidos

1. Validación
   - Formas de validar
2. Diseño por contrato
   - Correctitud
   - Especificación
3. Aserciones
   - Tipos de aserciones
   - Sintaxis Java
4. Verificación formal
   - Ternas de Hoare
   - Estados, asertos y expresiones

Verificación
Verificación formal

- Es un Conjunto de técnicas de comprobación formales.
  Permiten demostrar si un programa funciona correctamente.
- Programa que funciona correctamente
  Cumple con unas especificaciones dadas.
- Técnicas de comprobación
  - proceso de inferencia
  - Cada tipo de sentencia ejecutable posee una regla de inferencia
- Representación formal: Ternas de Hoare.

Verificación
Lógica de Hoare

Sistema formal desarrollado por C.A.R. Hoare (refinado por otros investigadores) Proporciona a una serie de reglas de inferencia para

La corrección de programas imperativos con el rigor de la lógica matemática.
La característica de esta lógica es la terna "Q S R", donde Q y R son predicados lógicos que deben cumplirse para que el programa S funcione.
Este método es la base del diseño de software por contrato.

Diseño por Contrato

Fórmula de Correctitud

Una Fórmula de Correctitud es una expresión lógica que dada una operación S y dos fórmulas lógicas P y Q, se cumple:

$$\{P\}\ C\ \{Q\}$$

Cualquier ejecución de C comenzando en un estado donde se cumple P, terminará en un estado donde se cumple Q

La **precondición** $\{P\}$ establece las propiedades que se tienen que cumplir cada vez que se llame a la operación

La **poscondición** $\{Q\}$ establece las propiedades que debe garantizar la operación cuando retorne

Representación formal
Ternas de Hoare

Sea C una parte del código, cualquier aserción $\{P\}$ se denomina precondición de C si $\{P\}$ sólo implica al estado inicial, cualquier aserción $\{Q\}$ se denomina postcondición si $\{Q\}$ sólo implica el estado final.

- Esta definición se representa como: $\{P\}$ C $\{Q\}$ y se denomina terna de Hoare.
- Ejemplo 1:$\{y \neq 0\}$ $x := 1/y$ $\{x = 1/y\}$.
- Ejemplo 2: A veces no existe precondición $\{\ \}$ $a := b$ $\{a = b\}$.

Siempre se obtienen estados que satisfacen la postcondición sin importarla precondición.
$\{\ \}$ representa la aserción vacía que se puede entender como "verdadero para todos los estados".

Correctitud
Código correcto

$$\{P\}\, C\, \{Q\}$$

es un código correcto si cada estado inicial posible que satisfaga $\{P\}$ da como resultado un estado final que satisface $\{Q\}$.

- Corrección parcial:
  $\{P\}\, C\, \{Q\}$ es parcialmente correcto si el estado final de C, cuando termina el programa , satisface $\{Q\}$ siempre que el estado inicial satisface $\{P\}$.

Correctitud
Código correcto

$$\{P\}\, C\, \{Q\}$$

es un código correcto si cada estado inicial posible que satisfaga $\{P\}$ da como resultado un estado final que satisface $\{Q\}$.

- Corrección parcial:
  $\{P\}\, C\, \{Q\}$ es parcialmente correcto si el estado final de C, cuando termina el programa , satisface $\{Q\}$ siempre que el estado inicial satisface $\{P\}$.

- Corrección total:
  Es parcialmente correcto y termina.

Asertos
Definición

Definición

Un aserto, predicado o aserción es una expresión lógica que involucra las variables del programa que usamos para expresar el comportamiento de dicho programa en ciertos momentos.
Las aserciones o asertos hacen referencia a un estado del sistema.

Asertos

Asertos débiles y fuertes

- Si una aserción A_1 es más fuerte que una aserción A_2
- Por lo tanto $A_1 \implies A_2$
- Todos los estados que satisfacen A_1 también satisfacen A_2 pero no viceversa.
- Un fortalecimiento de la aserción disminuye el número de estados que la pueden satisfacer
  - Más fuerte = más selectivo, más específico.
  - Más débil = menos selectivo, más general.

Asertos

Asertos débiles y fuertes

- A_1 es más fuerte que una aserción A_2, entonces $A_1 \implies A_2$
- Todos los estados que satisfacen A_1 también satisfacen A_2 pero no viceversa.
  - A_1 es $\{i > 1\}$, A_2 es $\{i > 0\}$, A_1 es más fuerte que A_2
- $\{i > 1\}$ es más fuerte que $\{i > 0\}$, los estados posibles que satisfacen la condición $\{i > 1\}$ también satisfacen que $\{i > 0\}$.
- Por lo tanto $\{i > 1\} \implies \{i > 0\}$.

Asertos
Asertos débiles y fuertes

Dados dos asertos A_1 y A_2

- A_1 es más fuerte que A_2
  - si todo estado que cumpla A_1 ha de cumplir también A_2;
  - dicho de otro modo $A_1 \Rightarrow A_2$;
  - el conjunto de los estados que satisfacen A_1 es un subconjunto de los estados que satisfacen A_2.
  - A_2 es más débil que A_1
  - no todo el conjunto de los estados que satisfacen A_2 satisfacen también a A_1.

Asertos
Ejemplos de asertos

Supongamos que se ha demostrado $\{A\}\, P\, \{B\}$, entonces:

- Si $A_1 \Rightarrow A$ (todos los estados que satisfacen A_1 se satisfacen en A);
  - entonces $\{A_1\}\, P\, \{B\}$ es verdadero;
- Si $B \Rightarrow B_1$
  - entonces $\{A\}\, P\, \{B_1\}$ es verdadero;

ejemplo

$\{y = 4\}$ es más fuerte que $\{y \neq 0\}$, por lo tanto:
Si se cumple la terna
$\{y \neq 0\}$ x := 1/y $\{x = 1/y\}$ se cumple también
$\{y = 4\}$ x := 1/y $\{x = 1/y\}$.

Asertos
Aserciones más fuertes

- Si una aserción A_1 es más fuerte que una aserción A_2 entonces todos los estados que satisfacen A_1 también satisfacen A_2 pero no viceversa.
- Un fortalecimiento de la aserción disminuye el número de estados que la pueden satisfacer.
  - Más fuerte $\Rightarrow$ más selectivo, más específico.
  - Más débil $\Rightarrow$ menos selectivo, más general.
- La aserción más débil es $\{\ \}$ que recoge todos los estados posibles. Constante lógico TRUE
- La aserción más fuerte será por tanto FALSE, ya que representa que ningún estado satisface dicha condición

Predicado
Logica de Primer Orden

LPO = lógica o cálculo de predicados

- Variables $\{x, yz, x_1\}$. su valor cambia durante el cómputo
- Constantes (valor fijo)
- funciones (operan sobre valores devolviendo otro valor)
- predicados (funciones que devuelven un valor booleano).

Asocia a cada fórmula un valor booleano dependiendo del estado de cómputo de las variables

Sintaxis

- Terminos: combinados forman una fórmula. Ejemplo: variable x, constante c, función $f(x_1, x_2, , x_n)$.
- Formulas: Conjunción de términos que se evalúa como lógico. Usa conectores y cuantificadores $\forall x, \exists x, \exists! x, \nexists x$.

Asertos
Conceptos

- Los asertos se indican encerrando una sentencia entre llaves $\{A\}$.
- Estado:
  conjunto de valores que toman en ese instante el conjunto de variables del problema.
- Precondiciones:Precondiciones: condiciones que deben satisfacer los datos de entrada .
- Poscondiciones: condiciones de salida aceptables como soluciones correctas del problema en cuestión.

Ejemplo

Programa que calcula la raiz cuadrada

- Precondición: Que el argumento de la función no sea negativo
- Postcondición: La raíz cuadrada de ese argumento.

Asertos
Estados de una variable

Definición

Definición: Si un programa usa n variables $(x_1, x_2, ..., x_n)$ el estado s es una tupla de valores $(X_1, X_2, ..., X_n)$ donde X_i es el valor de la variable x_i.

Ejemplo: Dado el estado

$s = (x, y) = (7, 8),$	estado previo
$x := 2 * y + 1$	sentencia
$s' = (x, y) = (17, 8)$	estado resultado

Ejemplo

Una **variable** se usa en un programa para describir una posición de memoria que puede contener valores diferentes en diferentes estados.

Una manera de describir un conjunto de estados es utilizando **fórmulas** del **cálculo de predicados**.

Asertos
Ejemplos de Estados de una variable

Es decir, es el predicado que solamente es satisfecho por los estados que pertenecen al conjunto S.

Ejemplo

$x := 2*y+1$. transforma el estado $s = (x, y)$ en el estado $s' = (2*y+1, y)$, los predicados $P(x, y)$ y $P'(x', y')$ tal que si $P(x, y)$ es verdadero en el estado s entonces $P'(x', y')$ será verdadero después de ejecutar la sentencia $x := 2 * y + 1$.

Ejemplo: Probar $x \leq 7$ después de ejecutar $x := 2 * y + 1$.

- $\{y \leq 3\}\, x := 2 * y + 1\, \{x \leq 7\}$
- $\{y \leq 3\}\, y\, \{x \leq 7\}$ se denominan aserciones.

Esto es verdadero si $2 * y + 1 \leq 7$ antes de ejecutar la sentencia, es decir $y \leq 3$.

Asertos
Estado inicial y final

- Un programa es una secuencia de sentencias que transforman el estado inicial en un estado final.
- El estado inicial es el estado anterior a la ejecución de un código.
- El estado final es el estado posterior a la ejecución de dicho código.
- El estado del sistema viene determinado por los valores de las variables en cada momento

Asertos
Especificación pre/post

- Un programa es una secuencia de sentencias que transforman el estado inicial en un estado final.
- El estado inicial es el estado anterior a la ejecución de un código.
- El estado final es el estado posterior a la ejecución de dicho código.
- El estado del sistema viene determinado por los valores de las variables en cada momento

Asertos
Tipos de Variable

Las variables que aparezcan en los asertos podrán ser de tipo:

- Variable ligada:
  - está vinculada a un cuantificador,
  - puede ser sustituida por cualquier otra sin que se modifique el significado de la expresión.
- Variable libre:
  - del programa: que denotan, en cada punto del mismo, el último valor que se les haya asignado.
  - Iniciales: se usan para denotar un valor que sólo se conoce en un punto diferente de la ejecución del programa.

Conviene evitar el uso del mismo nombre para variables del programa y ligadas

Asertos
Especificación Pre/post

- Al expresar, mediante un aserto, se está restringiendo el conjunto de estados en que se puede poner en marcha el programa con garantías de funcionamiento correcto.
- Este aserto que impone condiciones a los datos se denomina precondición. Cuanto más débil sea esta precondición, más útil será el programa, ya que podrá emplearse en más casos.
- El aserto que expresa las propiedades de los resultados del algoritmo se denomina postcondición.

Una especificación pre/post para un programa P se escribe:

$$\{P\}\ C\ \{Q\}$$

.

Asertos
Cuantificadores

Indica el conjunto de valores que permitimos tomar a la variable ligada α y que frecuentemente es un intervalo de los naturales

- $\sum \alpha$: dominio: E(α) -suma de los valores tomados expresión E
- $\prod \alpha$: dominio: E(α) -producto de los valores

- $\forall \alpha$: dominio: E(α) - todos los valores indicados en el dominio
- $\exists \alpha$: dominio: E(α) - existe un valor

- N α : dominio: E(α) - número de valores para lo que E es cierta
- max α : dominio: E(α) - maximo valor
- min α : dominio: E(α) - mínimo valor

Asertos
Ejemplos con cuantificadores

La variable a está declarada como un arreglo de enteros y α el índice entre 0 y $n-1$;

1. $\sum \alpha(0 \leq \alpha \leq n-1 \rightarrow a(\alpha))-$ entero resultante de sumar todos los elementos de a.
2. $\prod \alpha(0 \leq \alpha \leq n-1 \rightarrow a(\alpha))-$ entero resultante de multiplicar todos los elementos de a.
3. $\forall \alpha(0 \leq \alpha \leq n-1 \rightarrow a(\alpha) = 0)-$ vale *verdadero* si a tiene todos los elementos igual a 0, y falso en caso contrario.

Asertos
Ejemplos con cuantificadores

La variable *a* está declarada como un arreglo de enteros y α el índice entre 0 y $n-1$;

1. $\exists\alpha(0 \leq \alpha \leq n-1 \rightarrow a(\alpha) = 0)-$ vale *verdadero* si algún elemento de *a* es igual a 0.
2. $N\alpha(0 \leq \alpha \leq n-1 \rightarrow a(\alpha) = 0)-$ contar los elementos de *a* que son iguales a 0.
3. $max\ \alpha(0 \leq \alpha \leq n-1 \rightarrow a(\alpha))-$ corresponde al mayor valor que aparece en *a*.
4. $min\ \alpha(0 \leq \alpha \leq n-1 \rightarrow a(\alpha))-$ corresponde al menor valor que aparece en *a*.