



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Sviluppo software in gruppi di lavoro complessi¹

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Anno accademico 2022/23, II semestre

¹ © 2023 M. Monga. Creative Commons Attribuzione — Condividi allo stesso modo 4.0 Internazionale. <http://creativecommons.org/licenses/by-sa/4.0/deed.it>



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Lezione XIV: Contratti



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Meccanismi per monitorare l'aderenza alle specifiche

Il meccanismo base per monitorare/verificare l'aderenza di una implementazione alle specifiche (e ridurre gli *interface fault*):

Assertion

(1) a **logical expression** specifying a program state that must exist or a set of conditions that program variables must satisfy at a particular point during program execution. (2) a function or macro that complains loudly if a design assumption on which the code is based is not true.



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

assert (3)

NAME

assert - abort the program if assertion is false

SYNOPSIS

```
#include <assert.h>
```

```
void assert(scalar expression);
```

DESCRIPTION

If the macro `NDEBUG` was defined at the moment `<assert.h>` was last included, the macro `assert()` generates no code, and hence does nothing at all. Otherwise, the macro `assert()` prints an error message to standard error and terminates the program by calling `abort(3)` if expression is false (i.e., compares equal to zero).

CONFORMING TO

POSIX.1-2001, C89, C99. In C89, expression is required to be of type `int`.

BUGS

`assert()` is implemented as a macro; if the expression tested has side-effects, program behavior will be different depending on whether `NDEBUG` is defined. This may create Heisenbugs which go away when debugging is turned on.



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Ormai presente in quasi tutti i linguaggi nativo o nelle librerie standard:

Java `assert`

Python `assert`

PHP `assert`

Javascript `console.assert` (non in Explorer...)

...



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

È utile ragionare su “pattern” di asserzioni, spesso codificati in assertion languages/libraries.

D. S. Rosenblum, “Towards a Method of Programming with Assertions”, ICSE 1992 (Most influential paper award ICSE 2002).

Descrive un preprocessore (APP) per produrre asserzioni: il preprocessore lavora su speciali “commenti” `/*@ @*/`:

- `assume`
- `promise`
- `return`
- `assert`



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

```
int square_root(int x);
/*@
  assume x >= 0;
  return y where y >= 0;
  return y where y*y <= x
  &&& x < (y+1)*(y+1);
  @*/
```

```
void swap(int* x, int* y);
/*@
  assume x &&& y &&& x != y;
  promise *x == in *y;
  promise *y == in *x;
  @*/
void swap(int* x, int* y) {
  *x = *x + *y;
  *y = *x - *y;
  /*@ assert *y == in *x; @*/
  *x = *x - *y;
}
```



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

- Consistency between arguments
- Dependency of return value on arguments
- Effect on global state/Frame specifications
- The context in which a function is called
- Subrange membership of data/Enumeration membership of data
- Non-null pointers
- Condition of the else part of complex if (and switch)
- Consistency between related data
- Intermediate summary of processing



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Se usate estensivamente, le asserzioni possono costituire una vera e propria **specifica** delle componenti del sistema. L'idea della **progettazione per contratto** (B. Meyer, 1986) è che il linguaggio per descrivere specifiche e implementazioni è lo stesso: la specifica è parte integrante del codice del sistema. La specifica è parte del "contratto" secondo cui ciascun componente fornisce i propri servizi al resto del sistema.

125



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Che tipo di specifiche si usano?

$$\{P\}S\{Q\}$$

Ogni esecuzione di S che parta da uno stato che soddisfa la condizione P (pre-condizione) **termina** in uno stato che soddisfa la condizione Q (post-condizione). Ogni programma che termina è corretto se e solo se vale la proprietà precedente.

126



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

La tripla di Hoare $\{P\}S\{Q\}$ può diventare un **contratto** fra chi *implementa* (fornitore) S e chi *usa* (cliente) S

- L'implementatore di S si impegna a garantire Q in tutti gli stati che soddisfano P
- L'utilizzatore di S si impegna a chiedere il servizio in un stato che soddisfa P ed è certo che se S termina, si giungerà in uno stato in Q vale

Il lavoro dell'implementatore è particolarmente facile quando: Q è True (vera per ogni risultato!) o quando P è False (l'utilizzatore non riuscirà mai a portare il sistema in uno stato in cui tocchi fare qualcosa!). Weakest precondition (data Q) o strongest postcondition (data P) **determinano** il ruolo di una feature.

127



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Eiffel

Un linguaggio *object-oriented* che introduce i **contratti** nell'interfaccia delle classi. Il contratto di default per un metodo ("feature") F è $\{True\}F\{True\}$.

Dispensa: <https://homes.di.unimi.it/monga/lucidi2223/sinossi-eiffel.pdf>

```
feature
  decrement
    -- Decrease counter by one.
  require
    item > 0 -- pre-condition
  do
    item := item - 1 -- implementation
  ensure
    item = old item - 1 -- post-condition
end
```

128



Svigrosso

Monga

Asserzioni

Il modello di

Eiffel

Contratti

Eiffel

What & How

Contratti ed

ereditarietà

Eccezioni

- pre/post-condizioni sulle feature (`require`, `ensure`)
- Invarianti di classe (`invariant`)
- asserzioni (`check`)
- loop invariant
(`from .. invariant .. until .. variant .. loop .. end`)

129



Svigrosso

Monga

Asserzioni

Il modello di
Eiffel

Contratti

Eiffel

What & How

Contratti ed

ereditarietà

Eccezioni

- **invarianti di classe** sono condizioni che devono essere vere in ogni momento “critico”, ossia osservabile dall'esterno. In pratica e come se facessero parte di ogni pre- e post-condizione.
- è possibile avere un supporto run-time alle **violazioni**: se una condizione non vale viene sollevata un'eccezione
- L'eccezione porta il sistema nel precedente stato stabile ed è possibile
 - terminare con un fallimento
 - riprovare

130

```
class ROOT_TEST_STABLE_STATES
create make
feature {NONE}
  secret: BOOLEAN
feature {ANY}
  make -- root class cannot have preconditions
    -- require ok_pre("make")
    do
      print("Executing make\n")
      mycommand; secret := TRUE
      ensure ok_post("make")
    end
  mycommand
    require ok_pre("mycommand")
    do
      print("Executing mycommand\n")
      secret := FALSE; myother("1"); secret := TRUE
      -- But what happens if myother is a "client"?
      -- secret := FALSE; Current.myother("2"); secret := TRUE
      ensure ok_post("mycommand")
    end
  myother (s: STRING)
    require ok_pre("myother")
    do
      print("Executing myother " + s + "\n")
      ensure ok_post("myother")
    end
  ok_inv: BOOLEAN do print("Checking ok_inv!\n"); Result := True; end
  ok_pre (w: STRING): BOOLEAN do print("Checking ok_pre @ " + w + "\n"); Result := True; end
  ok_post (w: STRING): BOOLEAN do print("Checking ok_post @ " + w + "\n"); Result := True; end
invariant ok_inv
end
```

131



Svigrosso

Monga

Asserzioni

Il modello di
Eiffel

Contratti

Eiffel

What & How

Contratti ed

ereditarietà

Eccezioni

132



Spesso si scrivono le “stesse” cose due volte:

```
do
    balance := balance - x
ensure
    balance = old balance - x
```

- Implementazione e specifica

- How & What

Il client è responsabile delle precondizioni, il fornitore di postcondizioni e invarianti.



Il *principio di sostituzione di Liskov* stabilisce che, perché un oggetto di una classe derivata soddisfi la relazione *is-a*, ogni suo metodo:

- deve essere accessibile a pre-condizioni uguali o più deboli del metodo della superclasse;
- deve garantire post-condizioni uguali o più forti del metodo della superclasse;

Altrimenti il “figlio” non può essere sostituito al “padre” senza alterare il sistema.



Le due condizioni sono quindi:

$$PRE_{parent} \implies PRE_{derived} \quad (1)$$

$$POST_{derived} \implies POST_{parent} \quad (2)$$

- (1) in un programma corretto non può succedere che PRE_{parent} valga e $PRE_{derived}$ no; l'oggetto *evoluto* deve funzionare in ogni stato in cui funzionava l'originale: non può avere **obblighzioni** più stringenti, semmai più lasche.
- (2) in un programma corretto non può succedere che valga $POST_{derived}$ ma non $POST_{parent}$; un stato corretto dell'oggetto *evoluto* deve essere corretto anche quando ci si attende i **benefici** dell'originale.



Un modo per garantire che le condizioni (1) e (2) siano automaticamente vere consiste nell'assumere implicitamente che, se la classe evoluta specifica esplicitamente una precondizione P e una postcondizione Q , le reali pre- e post-condizioni siano:

$$PRE_{derived} = PRE_{parent} \vee P \quad (3) \quad PRE_{parent} \implies PRE_{derived}$$

$$POST_{derived} = POST_{parent} \wedge Q \quad (4) \quad POST_{derived} \implies POST_{parent}$$

In Eiffel: `require else` e `ensure then`

Contratti "astratti"



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

```
extend (x: G)
-- Add `x' at end of list.
require
  space_available: not full
deferred
ensure
  one_more:
    count = old count + 1
end

full: BOOLEAN
-- Is representation full?
-- (Default: no)
do
  Result := False
end
```

Stronger precondition... ma weaker (uguali in realtà) in astratto

```
full: BOOLEAN
-- Is representation full?
-- (Answer: if and only if
-- number of items is equal
-- to capacity)
do
  Result := (count = capacity)
end
```

137

Problema: i parametri...



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

- Animale mangia Cibo (is_a Cosa)
- Mucca (is_a Animale) mangia Erba (is_a Cibo)

Ma questa **covarianza** è contraria al principio di Liskov perché restringe le precondizioni. La controvarianza (Mucca mangia Cosa, Sather) e l'invarianza (Mucca mangia Cibo, Java) vanno bene.

Eiffel invece è covariante... (il che, impedendo un controllo di conformità statico, introduce parecchie complicazioni ~> CATcall, run time type identification...).

138

Treatmento delle situazioni anomale



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Nel modello di Eiffel hanno un ruolo importante le eccezioni, che vengono trattate in un modo differente da quello dei più diffusi linguaggi di programmazione (Ada-like).

Exception

An exception is a run-time event that may cause a routine call to fail (**contract violation**). A failure of a routine causes an exception in its caller.

139

Una discrepanza fra contratto e stato *run-time*



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Sicuramente c'è un difetto.

- Se il contratto è corretto:
 - violazione delle precondizioni: responsabile è il *client*
 - violazione delle postcondizioni o invarianti: responsabile è l'implementatore

140

Il trattamento delle eccezioni in Eiffel



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Due modalità:

- 1 **Failure** (organized panic): clean up the environment, terminate the call and report failure to the caller.
- 2 **Retry**: attempt to change the conditions that led to the exception and to execute the routine again from the start.

Per trattare il secondo caso, Eiffel introduce il costrutto `rescue/retry`. Se il corpo del 'rescue' non fa 'retry', si ha un failure.

141

Esempio



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

```
make_with_current_time
local
f: RAW_FILE
t: INTEGER
do
create f.make_open_temporary
t := f.date.integer_remainder (24*60*60)
set_seconds (t.integer_remainder (60))
set_minutes (t.integer_remainder (60*60).integer_quotient (60))
t := t - t.integer_remainder (60*60)
set_hours(t.integer_quotient (60*60))
f.delete
end
```

142

Esempio



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

```
make_with_current_time
local
f: RAW_FILE
t: INTEGER
do
create f.make_open_temporary
t := f.date.integer_remainder (24*60*60)
set_seconds (t.integer_remainder (60))
set_minutes (t.integer_remainder (60*60).integer_quotient (60))
t := t - t.integer_remainder (60*60)
set_hours(t.integer_quotient (60*60))
f.delete
rescue
make (0, 0)
end
```

143

Esempio



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

```
div (num: REAL, denom: REAL): REAL
require
denom /= 0
deferred

quasi_inverse (x: REAL): REAL
-- div(1, x) if possible, otherwise 0
local
division_tried: BOOLEAN
do
if not division_tried then
Result := div (1, x)
else
Result := 0
end
rescue
division_tried := True
retry
end
```

144



Svigruppo

Monga

Asserzioni

Il modello di Eiffel

Contratti

Eiffel

What & How

Contratti ed ereditarietà

Eccezioni

Per ogni feature (pubblica) f :

- $\{PRE_f \wedge INV\} body_f \{POST_f \wedge INV\}$
- $\{True\} rescue_f \{INV\}$
- $\{True\} retry_f \{INV \wedge PRE_f\}$