Design-by-Contract Approach

Hernán Melgratti

ICC University of Buenos Aires-Conicet

18 February 2020 @ Pisa

Design-by-Contract ¹

Basics

- ▶ To specify the constraints that govern the design and correct use of a class
- ▶ Contract:
 - Class invariant: assertions about the state of an object that hold before and after each method call
 - Preconditions: assertions about the state of the object and the argument values that must hold prior to invoking the method
 - Postconditions: assertions about the state of the object after the execution of a method

¹Bertrand Meyer. Applying Design by Contract. In Computer IEEE, vol. 25, no. 10, October 1992

Example

Bank Account

- ▶ Property: balance
- ► Operations: deposit(int amt), withdraw(int amt)
- ▶ Invariant: balance > 0
- deposit(int n):
 - ▶ **pre**: *n* > 0
 - post: balance' = balance + n
- withdraw(int n):
 - ▶ pre: n < balance</p>
 - ▶ post: balance' = balance n

Interpretation

- ▶ Precondition: an obligation for the client and a guarantee for the supplier
- ▶ Postcondition: an obligation for the supplier and a guarantee for the client
- ▶ Invariant: a property that is **assumed on entry** and **guaranteed on completion**

Implementation

- ▶ The code is enriched with a specification of the contract
- ▶ A run-time mechanism monitors the satisfaction of the contract
 - 1. When a client invokes an method, the precondition is checked and an **exception is** raised if the precondition is violated
 - the client is blamed
 - 2. The provider executes the invoked code
 - After completion, the postcondition is evaluated and an exception is raised if the postcondition is violated
 - the provider is blamed

Contracts and Higher-order functions

$\mathsf{filter}\;((\mathsf{int}\to\mathsf{bool})\;\mathsf{pred}\;):([\mathsf{int}]\to[\mathsf{int}])$

- it receives a predicate to check whether an integer is an even number, and
- ▶ it returns a function that allows to filter the elements of a list that are even

Its contract could be:

- ▶ pre: $\forall x : int.(x \mod 2 = 0) \iff pred x$
- ▶ post: $\forall x : int.x \in (filter pred) \ ls \iff (x \in ls \& x \mod 2 = 0)$

Issues

- Checking of pre- and postconditions
 - we cannot check whether pred satisfies pre when filter is invoked
 - we cannot check if (filter pred) satisfies post on return
- ▶ Blame assignment:
 - if (filter pred) violates the postcondition, it may be because of pred
 - filter may use pred as a parameter when invoking auxiliary functions (and violates the contracts of auxiliary functions)

- An extension of Programming Computable Functions (PCF) with contracts for higher-order functions
- ▶ Expressions can be decorated with contracts that link a client and a provider
- ► Contracts are evaluated only over values of basic types (not over functions)
- When a contract is violated, blame is assigned to either the client or the provider

²Robert Bruce Findler, Matthias Felleisen: ICFP 2002: Contracts for higher-order functions.

³Christos Dimoulas, Robert Bruce Findler, Cormac Flanagan, Matthias Felleisen: Correct blame for contracts: no more scapegoating. POPL 2011.

Programming Computable Functions (PCF)

Syntax

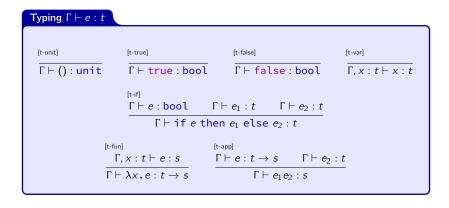
```
Types t ::= b \mid t \rightarrow t

b := int | bool | unit

Expression e ::= v \mid x \mid e_1e_2 \mid if e then e_1 else e_2 \mid \dots ( number expr)

Value v, w ::= () | true | false | \dots | \lambda x \cdot e
```

Programming Computable Functions (PCF)



Programming Computable Functions (PCF)

```
Semantics e \rightarrow e
                                        [if-true]
                                         if true then e_1 else e_2 \rightarrow e_1
                                       [if-false]
                                        if false then e_1 else e_2 \rightarrow e_2
                                                    [beta]
                                                    (\lambda x.e)v \rightarrow e\{v/x\}
                                                        [context]
                                                            e_1 \rightarrow e_2
                                                         \overline{\mathscr{E}[e_1] \to \mathscr{E}[e_2]}
```

$$\mathscr{E}$$
 ::= [] | $\mathscr{E}e$ | $v\mathscr{E}$ | if \mathscr{E} then e_1 else e_2

Programming Computable Functions (PCF) + Contracts

$mon^{k,l}(\kappa,e)$

- ightharpoonup Contract κ mediates the interaction between e (provider) and its context (client)
- \blacktriangleright any value that flows between e and its context is monitored for conformance with κ
- k and I are the blame labels for the two parties to the contract

error¹

▶ blame is assigned to *I*

Programming Computable Functions (PCF) + Contracts

flat(e)

- ► A contract for an expression of a basic type unit, bool, unit, ...
- e is a predicate (for values of a basic type)

$$flat(\lambda x.x \ge 0)$$

$\kappa_1 \mapsto \kappa_2$

- ▶ A contract for a function
- ightharpoonup κ_1 is the contract for the domain (the precondition)
- ightharpoonup is the contract for the codomain (the postcondition)

$$flat(\lambda x.x \ge 0) \mapsto flat(\lambda x.x \le 0)$$

Programming Computable Functions (PCF) + Contracts

```
Types t ::= b \mid t \rightarrow t \mid \operatorname{con}(t)

b := \operatorname{int} \mid \operatorname{bool} \mid \operatorname{unit}

Contracts \kappa ::= \operatorname{flat}(e) \mid \kappa \mapsto \kappa

Expression e ::= v \mid x \mid e_1 e_2 \mid \operatorname{if} e \operatorname{then} e_1 \operatorname{else} e_2 \mid \dots (\operatorname{number} \operatorname{expr}) \mid \operatorname{mon}^{l,l}(\kappa,e) \mid \operatorname{error}^l

Value v,w ::= () \mid \operatorname{true} \mid \operatorname{false} \mid \dots \mid \lambda x \cdot e
```