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
  - **Pre**conditions: assertions about the state of the object and the argument values that must hold prior to invoking the method
  - **Post**conditions: assertions about the state of the object after the execution of a method

[1] Bertrand 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**: $\mathtt{balance}' = \mathtt{balance} + \mathtt{n}$
- `withdraw(int n)`:
  - **pre**: $\mathtt{n} < \mathtt{balance}$
  - **post**: $\mathtt{balance}' = \mathtt{balance} - \mathtt{n}$

Interpretation

- **Pre**condition: an **obligation for the client** and a **guarantee for the supplier**
- **Post**condition: 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
  3. 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

filter ((int → bool) pred) : ([int] → [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**: $\forall x : \texttt{int}.(x \bmod 2 = 0) \iff \texttt{pred}\ x$
- **post**: $\forall x : \texttt{int}.x \in (\texttt{filter}\ \textit{pred})\ \textit{ls} \iff (x \in \textit{ls}\ \&\ x \bmod 2 = 0)$

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)

λ^{con} [2] [3]

- 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

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

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

Programming Computable Functions (PCF)

Syntax

Types	t	$::=$	$b \mid t \to t$
	b	$:=$	$\texttt{int} \mid \texttt{bool} \mid \texttt{unit}$
Expression	e	$::=$	$v \mid x \mid e_1 e_2 \mid \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \ldots$ (*number expr*)
Value	v, w	$::=$	$() \mid \texttt{true} \mid \texttt{false} \mid \ldots$
			$\mid \lambda x . e$

Programming Computable Functions (PCF)

Typing $\Gamma \vdash e : t$

[t-unit]
$$\frac{}{\Gamma \vdash () : \mathtt{unit}}$$

[t-true]
$$\frac{}{\Gamma \vdash \mathtt{true} : \mathtt{bool}}$$

[t-false]
$$\frac{}{\Gamma \vdash \mathtt{false} : \mathtt{bool}}$$

[t-var]
$$\frac{}{\Gamma, x : t \vdash x : t}$$

[t-if]
$$\frac{\Gamma \vdash e : \mathtt{bool} \qquad \Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : t}{\Gamma \vdash \mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : t}$$

[t-fun]
$$\frac{\Gamma, x : t \vdash e : s}{\Gamma \vdash \lambda x . e : t \to s}$$

[t-app]
$$\frac{\Gamma \vdash e : t \to s \qquad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 e_2 : s}$$

Programming Computable Functions (PCF)

Semantics $e \rightarrow e$

[if-true]

$$\frac{}{\texttt{if true then } e_1 \texttt{ else } e_2 \rightarrow e_1}$$

[if-false]

$$\frac{}{\texttt{if false then } e_1 \texttt{ else } e_2 \rightarrow e_2}$$

[beta]

$$\frac{}{(\lambda x.\, e)v \rightarrow e\{v/x\}}$$

[context]

$$\frac{e_1 \rightarrow e_2}{\mathscr{E}[e_1] \rightarrow \mathscr{E}[e_2]}$$

$$\mathscr{E} \quad ::= \quad [\,] \mid \mathscr{E}e \mid v\mathscr{E} \mid \texttt{if } \mathscr{E} \texttt{ then } e_1 \texttt{ else } e_2$$

Programming Computable Functions (PCF) + Contracts

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

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

$\texttt{error}^l$

- blame is assigned to l

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)

$$\mathsf{flat}(\lambda x.x \geq 0)$$

$\kappa_1 \mapsto \kappa_2$

- A contract for a function
- κ_1 is the contract for the domain (the precondition)
- κ_2 is the contract for the codomain (the postcondition)

$$\mathsf{flat}(\lambda x.x \geq 0) \mapsto \mathsf{flat}(\lambda x.x \leq 0)$$

Programming Computable Functions (PCF) + Contracts

Syntax

Types	t	$::=$	$b \mid t \to t \mid \mathtt{con}(t)$
	b	$:=$	$\mathtt{int} \mid \mathtt{bool} \mid \mathtt{unit}$
Contracts	κ	$::=$	$\mathtt{flat}(e) \mid \kappa \mapsto \kappa$
Expression	e	$::=$	$v \mid x \mid e_1 e_2 \mid \mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \mid \ldots$ (*number expr*)
		$\mid$	$\mathtt{mon}^{l,l}(\kappa, e) \mid \mathtt{error}^{l}$
Value	v, w	$::=$	$() \mid \mathtt{true} \mid \mathtt{false} \mid \ldots$
		$\mid$	$\lambda x.e$