Design-by-Contract Approach

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: $\texttt{balance} > \texttt{0}$
- `deposit(int amt)`:
  - **pre**: $\texttt{amt} > \texttt{0}$
  - **post**: $\texttt{balance}' = \texttt{balance} + \texttt{amt}$
- `withdraw(int n)`:
  - **pre**: $0 < \texttt{amt} < \texttt{balance}$
  - **post**: $\texttt{balance}' = \texttt{balance} - \texttt{amt}$

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**: $\forall x : \texttt{int}.x \in (\texttt{filter}\ pred)\ ls \iff (x \in\ ls\ \&\ x \bmod 2 = 0)$

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

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

Programming Computable Functions (PCF)

Typing $\Gamma \vdash e : t$

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

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

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

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

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

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

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

Programming Computable Functions (PCF)

Semantics $e \to e$

[if-true]

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

[if-false]

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

[beta]

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

[context]

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

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

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

$\texttt{flat}(e)$

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

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

$\kappa_1 \mapsto \kappa_2$

- 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)

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

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	$::=$	$() \mid \mathtt{true} \mid \mathtt{false} \mid \ldots$
		$\mid$	$\lambda x . e$

Programming Computable Functions (PCF) + Contracts

Example

$$
\begin{aligned}
\mathtt{ge} &= \lambda x\,.\,x \geq 0 \\
\mathtt{le} &= \lambda x\,.\,x \leq 0 \\
\mathtt{op} &= \lambda x\,.\,x * (-1) \\
\mathtt{op}_{\mathtt{mon}} &= \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{ge}) \mapsto \mathtt{flat}(\mathtt{le}), \mathtt{op})
\end{aligned}
$$

Programming Computable Functions (PCF)+ Contracts

Typing $\Gamma \vdash e : t$

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

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

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

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

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

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

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

Typing for contracts $\Gamma \vdash e : t$

[t-flat]
$$\frac{\Gamma \vdash e : o \to \texttt{bool}}{\Gamma \vdash \texttt{flat}(e) : \mathrm{con}(o)}$$

[t-contract]
$$\frac{\Gamma \vdash \kappa_1 : \mathrm{con}(t_1) \qquad \Gamma \vdash \kappa_2 : \mathrm{con}(t_2)}{\Gamma \vdash \kappa_1 \mapsto \kappa_2 : \mathrm{con}(t_1 \to t_2)}$$

[t-mon]
$$\frac{\Gamma \vdash \kappa : \mathrm{con}(t) \qquad \Gamma \vdash e : t}{\Gamma \vdash \mathrm{mon}^{k,l}(\kappa, e) : t}$$

[t-error]
$$\frac{}{\Gamma \vdash \texttt{error}^{l} : t}$$

Programming Computable Functions (PCF) + Contracts

Semantics $e \to e$

[if-true]

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

[if-false]

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

[beta]

$$\frac{}{(\lambda x\texttt{.}e)v \to e\{v/x\}}$$

[context]

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

[flat]

$$\frac{}{\texttt{mon}^{k,l}(\texttt{flat}(e), v) \to \texttt{if}\ ev\ \texttt{then}\ v\ \texttt{else}\ \texttt{error}^k}$$

[func]

$$\frac{}{\texttt{mon}^{k,l}(\kappa_1 \mapsto \kappa_2, v) \to \lambda x\texttt{.mon}^{k,l}(\kappa_2, v\ \texttt{mon}^{l,k}(\kappa_1, x))}$$

[context-error]

$$\frac{}{\mathscr{E}[\texttt{error}^k] \to \texttt{error}^k}$$

$$\mathscr{E} \quad ::= \quad [\,] \mid \mathscr{E}e \mid v\mathscr{E} \mid \ \mid \texttt{if}\ \mathscr{E}\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \texttt{mon}^{l,l}(\kappa, \mathscr{E})$$

Programming Computable Functions (PCF) + Contracts

Example

$$\begin{aligned}
\mathtt{ge} &= \lambda x.x \geq 0 \\
\mathtt{le} &= \lambda x.x \leq 0 \\
\mathtt{op} &= \lambda x.x * (-1) \\
\mathtt{op}_{\mathtt{mon}} &= \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{ge}) \mapsto \mathtt{flat}(\mathtt{le}), \mathtt{op})
\end{aligned}$$

$$\mathtt{op}_{\mathtt{mon}} \to \lambda x.\mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ \mathtt{mon}^{l,k}(\mathtt{flat}(\mathtt{ge}), x))$$

$\mathtt{op}_{\mathtt{mon}}$ 1

$$\begin{aligned}
\mathtt{op}_{\mathtt{mon}}\ 1 &\to (\lambda x.\mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ \mathtt{mon}^{l,k}(\mathtt{flat}(\mathtt{ge}), x)))\ 1 \\
&\to \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ \mathtt{mon}^{l,k}(\mathtt{flat}(\mathtt{ge}), 1)) \\
&\to \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ \mathtt{if}\ \mathtt{ge}\ 1\ \mathtt{then}\ 1\ \mathtt{else}\ \mathtt{error}^{l}) \\
&\to \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ \mathtt{if}\ 1 \geq 0\ \mathtt{then}\ 1\ \mathtt{else}\ \mathtt{error}^{l}) \\
&\to \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ \mathtt{if}\ \mathtt{true}\ \mathtt{then}\ 1\ \mathtt{else}\ \mathtt{error}^{l}) \\
&\to \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ 1) \\
&\to \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), 1 * (-1)) \\
&\to \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), (-1)) \\
&\to \mathtt{if}\ \mathtt{le}\ (-1)\ \mathtt{then}\ (-1)\ \mathtt{else}\ \mathtt{error}^{k} \\
&\to \mathtt{if}\ (-1) \leq 0\ \mathtt{then}\ (-1)\ \mathtt{else}\ \mathtt{error}^{k} \\
&\to \mathtt{if}\ \mathtt{true}\ \mathtt{then}\ (-1)\ \mathtt{else}\ \mathtt{error}^{k} \\
&\to (-1)
\end{aligned}$$

Programming Computable Functions (PCF) + Contracts

Example

$$\begin{aligned}
\texttt{ge} &= \lambda x.x \geq 0 \\
\texttt{le} &= \lambda x.x \leq 0 \\
\texttt{op} &= \lambda x.x * (-1) \\
\texttt{op}_{\texttt{mon}} &= \texttt{mon}^{k,l}(\texttt{flat}(\texttt{ge}) \mapsto \texttt{flat}(\texttt{le}), \texttt{op})
\end{aligned}$$

$$\texttt{op}_{\texttt{mon}} \to \lambda x.\texttt{mon}^{k,l}(\texttt{flat}(\texttt{le}), \texttt{op}\ \texttt{mon}^{l,k}(\texttt{flat}(\texttt{ge}), x))$$

$\texttt{op}_{\texttt{mon}}(-1)$

$$\begin{aligned}
\texttt{op}_{\texttt{mon}}(-1) &\to (\lambda x.\texttt{mon}^{k,l}(\texttt{flat}(\texttt{le}), \texttt{op}\ \texttt{mon}^{l,k}(\texttt{flat}(\texttt{ge}), x)))\ (-1) \\
&\to \texttt{mon}^{k,l}(\texttt{flat}(\texttt{le}), \texttt{op}\ \texttt{mon}^{l,k}(\texttt{flat}(\texttt{ge}), (-1))) \\
&\to \texttt{mon}^{k,l}(\texttt{flat}(\texttt{le}), \texttt{op}\ \texttt{if}\ \texttt{ge}\ (-1)\ \texttt{then}\ (-1)\ \texttt{else}\ \texttt{error}^{l}) \\
&\to \texttt{mon}^{k,l}(\texttt{flat}(\texttt{le}), \texttt{op}\ \texttt{if}\ (-1) \geq 0\ \texttt{then}\ 1\ \texttt{else}\ \texttt{error}^{l}) \\
&\to \texttt{mon}^{k,l}(\texttt{flat}(\texttt{le}), \texttt{op}\ \texttt{if}\ \texttt{false}\ \texttt{then}\ 1\ \texttt{else}\ \texttt{error}^{l}) \\
&\to \texttt{mon}^{k,l}(\texttt{flat}(\texttt{le}), \texttt{error}^{l}) \\
&\to \texttt{error}^{l}
\end{aligned}$$

The blame is assigned to the **client**

Programming Computable Functions (PCF) + Contracts

Example

$$\begin{aligned}
\mathtt{ge} &= \lambda x.\, x \geq 0 \\
\mathtt{le} &= \lambda x.\, x \leq 0 \\
\mathtt{op} &= \lambda x.\, x \\
\mathtt{op}_{\mathtt{mon}} &= \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{ge}) \mapsto \mathtt{flat}(\mathtt{le}), \mathtt{op})
\end{aligned}$$

$$\mathtt{op}_{\mathtt{mon}} \to \lambda x.\, \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ \mathtt{mon}^{l,k}(\mathtt{flat}(\mathtt{ge}), x))$$

$\mathtt{op}_{\mathtt{mon}}\,1$

$$\begin{aligned}
\mathtt{op}_{\mathtt{mon}}\,1 &\to (\lambda x.\, \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ \mathtt{mon}^{l,k}(\mathtt{flat}(\mathtt{ge}), x)))\ 1 \\
&\to^{*} \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), \mathtt{op}\ 1) \\
&\to \mathtt{mon}^{k,l}(\mathtt{flat}(\mathtt{le}), 1) \\
&\to \mathtt{if}\ \mathtt{le}\ 1\ \mathtt{then}\ 1\ \mathtt{else}\ \mathtt{error}^{k} \\
&\to \mathtt{if}\ 1 \leq 0\ \mathtt{then}\ 1\ \mathtt{else}\ \mathtt{error}^{k} \\
&\to \mathtt{if}\ \mathtt{false}\ \mathtt{then}\ 1\ \mathtt{else}\ \mathtt{error}^{k} \\
&\to \mathtt{error}^{k}
\end{aligned}$$

The blame is assigned to the **provider**

Dependent contracts

$$\kappa := \ldots \mid \kappa \overset{d}{\mapsto} (\lambda x\,.\,\kappa)$$

Example

$$\begin{aligned}
\texttt{succ} &= \lambda x\,.\,x + 1 \\
\kappa &= \texttt{flat}(\lambda x\,.\,\texttt{true}) \overset{d}{\mapsto} (\lambda x\,.\,\texttt{flat}(\lambda y\,.\,y > x)) \\
\texttt{succ}_{\texttt{mon}} &= \texttt{mon}^{k,l}(\kappa, \texttt{succ})
\end{aligned}$$

Dependent contracts

Typing for contracts $\Gamma \vdash e : t$

[t-contract]

$$\frac{\Gamma \vdash \kappa_1 : \mathsf{con}(t_1) \quad \Gamma \vdash \kappa_2 : t_1 \to \mathsf{con}(t_2)}{\Gamma \vdash \kappa_1 \overset{d}{\mapsto} \kappa_2 : \mathsf{con}(t_1 \to t_2)}$$

Semantics $e \to e$

[d-func-lax]

$$\frac{}{\mathsf{mon}^{k,l}(\kappa_1 \overset{d}{\mapsto} (\lambda x.\kappa_2), v) \to \lambda x.\mathsf{mon}^{k,l}(\kappa_2, v\ \mathsf{mon}^{l,k}(\kappa_1, x))}$$

Dependent Contract

Example

$$\begin{aligned}
\mathtt{succ} &= \lambda x.\, x + 1 \\
\kappa &= \mathtt{flat}(\lambda x.\, \mathtt{true}) \overset{d}{\mapsto} (\lambda x.\, \mathtt{flat}(\lambda y.\, y > x)) \\
\mathtt{succ}_{\mathtt{mon}} &= \mathtt{mon}^{k,l}(\kappa, \mathtt{succ})
\end{aligned}$$

$$\mathtt{succ}_{\mathtt{mon}} \to \lambda x.\, \mathtt{mon}^{k,l}(\mathtt{flat}(\lambda y.\, y > x), \mathtt{succ}\ \mathtt{mon}^{l,k}(\mathtt{flat}(\lambda x.\, \mathtt{true}), x))$$

$\mathtt{succ}_{\mathtt{mon}}$ 2

$$\begin{aligned}
\mathtt{succ}_{\mathtt{mon}} &\to (\lambda x.\, \mathtt{mon}^{k,l}(\mathtt{flat}(\lambda y.\, y > x), \mathtt{succ}\ \mathtt{mon}^{l,k}(\mathtt{flat}(\lambda x.\, \mathtt{true}), x)))\ 2 \\
&\to \mathtt{mon}^{k,l}(\mathtt{flat}(\lambda y.\, y > 2), \mathtt{succ}\ \mathtt{mon}^{l,k}(\mathtt{flat}(\lambda x.\, \mathtt{true}), 2)) \\
&\to^* \mathtt{mon}^{k,l}(\mathtt{flat}(\lambda y.\, y > 2), \mathtt{succ}\ 2) \\
&\to^* \mathtt{mon}^{k,l}(\mathtt{flat}(\lambda y.\, y > 2), 3) \\
&\to^* 3
\end{aligned}$$

Results

Lemma (Preservation)

If $\emptyset \vdash e : t$ *and* $e \to e'$, *then* $\emptyset \vdash e' : t$

Lemma (Type Soundness)

If $\emptyset \vdash e : t$ *then either:*

- $e \to^* v$, *or*
- $e \to^* \texttt{error}^l$.