



Last update: 28 March 2007

Software Architecture

Bertrand Meyer

ETH Zurich, March-July 2007

Exercise:

Abstract Data Types by Example
(Bernd Schoeller)



TYPE DECLARATION

FUNCTION DECLARATION

PRECONDITIONS

AXIOMS



TYPES

DATABASE, LIST[X], RELATION[X, Y]

X and **Y** are formal type parameters

All types provide comparison:

= : T × T → BOOLEAN



TYPES

DATABASE, LIST[X], RELATION[X, Y]

X and *Y* are formal type parameters

All types provide comparison:

= : T × T → BOOLEAN

Some predefined types (and related functions):

BOOLEAN (*True*, *False*, **and**, **or**, **not**, **implies**, etc.)

INTEGER, NATURAL (*Number*, **+**, **-**, *****, **/**, etc.)

(in prefix/infix notation)



FUNCTIONS

abs: INTEGER \rightarrow INTEGER

sqrt: FLOAT \rightarrow FLOAT

count: LIST[G] \rightarrow INTEGER

first: LIST[G] \rightarrow G

has: LIST[G] \times G \rightarrow BOOLEAN

sum: LIST[INTEGER] \rightarrow INTEGER

pi: FLOAT

empty_list: LIST[G]

\rightarrow : Total Function, \rightarrow : Partial Function

Well-formed, well-typed terms



`item (new_stack)`

`item (put (new_stack, 3)) = 2`

`sqrt (sqrt (4.1))`

`sqrt (8 - 10)`

`i_th (extend (new_list, "Hallo"), 3)`

`item (put (put (remove (put (new_stack, 4)), 3), 2)) ≠ item (put (new_stack, 1+2))`

Preconditions



Terms that have to be *equivalent* (=) to **True** to allow the *application* of the function:

PRECONDITIONS (\forall p: **PERSON**, s: **STACK[G]**)

spouse (p) **require** **is_married** (p)

item (s) **require** **not empty** (s)

Preconditions



Terms that have to be *equivalent* (=) to **True** to allow the *application* of the function:

PRECONDITIONS (\forall p: **PERSON**, s: **STACK[G]**)

spouse (p) **require** **is_married** (p)
item (s) **require not** **empty** (s)

Alternative notation (used in the lecture):

PRECONDITIONS

spouse (p: **PERSON**) **require** **is_married** (p)
item (s: **STACK[G]**) **require not** **empty** (s)

Correctness of terms



“A term of which we can prove that all arguments to ADT functions satisfy the preconditions (if any)”

Examples: correct terms



item (new_stack) Incorrect

item (put (new_stack, 3)) = 2 OK

sqrt (sqrt (4.1)) OK

sqrt (8 - 10) Incorrect

i_th (extend (new_list, "Hallo"), 3) Incorrect

item (put (put (remove (put (new_stack, 4)), 3), 2)) \neq item (put (new_stack, 1+2)) OK



Axioms describe the *equivalence* ($=$, \neq) of *terms*.

AXIOMS (\forall p: PERSON, s: STACK[G], x: G, i: NATURAL)

spouse (spouse (p)) = p

spouse (p) \neq p

item (put (s,x)) = x

remove (put (s,x)) = s

double (succ (i)) = succ (succ (double (i)))

remove_n (i+1, put (s,x)) = remove_n (i,s)

Or

remove_n (i, put (s,x)) = remove_n (i-1,s) *iff* $i > 0$

TABLE (mapping, relation)



TYPES

TABLE[K,V]

FUNCTIONS

new : TABLE[K,V]

put : TABLE[K,V] \times K \times V \rightarrow TABLE[K,V]

remove : TABLE[K,V] \times K \rightarrow TABLE[K,V]

item : TABLE[K,V] \times K \rightarrow V

has_key : TABLE[K,V] \times K \rightarrow BOOLEAN

Partial functions of TABLE



TYPES

$\text{TABLE}[K, V]$

FUNCTIONS

$\text{new} : \text{TABLE}[K, V]$

$\text{put} : \text{TABLE}[K, V] \times K \times V \rightarrow \text{TABLE}[K, V]$ (overwrite)

$\text{remove} : \text{TABLE}[K, V] \times K \rightarrow \text{TABLE}[K, V]$

$\text{item} : \text{TABLE}[K, V] \times K \rightarrow V$

$\text{has_key} : \text{TABLE}[K, V] \times K \rightarrow \text{BOOLEAN}$

Table: preconditions



PRECONDITIONS ($\forall t: \text{TABLE}[K, V], k: K$)

remove (t, k) **require** **has_key** (t,k)

(P1)

item (t, k) **require** **has_key** (t,k)

(P2)

Table: axioms for has_key



AXIOMS ($\forall t: \text{TABLE}[K, V], k, k': K, v: V$)

`has_key (new, k) = False` (AH1)

`has_key (put (t, k, v), k) = True` (AH2)

`has_key (put (t, k', v), k) = has_key (t, k) iff $k \neq k'$` (AH3)

Table: axioms for item



AXIOMS ($\forall t: \text{TABLE}[K, V], k, k': K, v: V$)

`item` (`put` (`t`, `k`, `v`), `k`) = `v` (A1)

`item` (`put` (`t`, `k'`, `v`), `k`) = `item` (`t`, `k`) *iff* $k \neq k'$ (A2)

Table: axioms for remove



AXIOMS ($\forall t: \text{TABLE}[K, V], k, k': K, v: V$)

$\text{remove}(\text{put}(t, k', v), k) = \text{put}(\text{remove}(t, k), k', v)$ *iff* $k \neq k'$
(AR1)

$\text{remove}(\text{put}(t, k, v), k) = t$
(AR2)

Table: axioms for remove



AXIOMS ($\forall t: \text{TABLE}[K, V], k, k': K, v: V$)

$\text{remove}(\text{put}(t, k', v), k) = \text{put}(\text{remove}(t, k), k', v)$ *iff* $k \neq k'$
(AR1)

$\text{remove}(\text{put}(t, k, v), k) = t$
(AR2)



Problems with remove



Example: `TABLE[STRING,INTEGER]`

`remove (put (put (put (new, "foo", 1), "foo", 2), "bar", 3), "foo")`

"foo" should be removed: {"bar",3}

Problems with remove



Example: `TABLE[STRING,INTEGER]`

`remove (put (put (put (new, "foo", 1), "foo", 2), "bar", 3), "foo")`

“foo” should be removed: {("bar",3)}

= (AR1)

`put (remove (put (put (new, "foo", 1), "foo", 2), "foo"), "bar", 3)`

(as “foo” and “bar” are not the same)

Problems with remove



Example: `TABLE[STRING,INTEGER]`

`remove (put (put (put (new, "foo", 1), "foo", 2), "bar", 3), "foo")`

"foo" should be removed: `{("bar",3)}`

= (AR1)

`put (remove (put (put (new, "foo", 1), "foo", 2), "foo"), "bar", 3)`

= (AR2)

`put (put (new, "foo", 1), "bar", 3)`

result is: `{("foo",1),("bar",3)}`
`remove` is really "undo last put"

Table: axioms for remove (corrected)



AXIOMS ($\forall t: \text{TABLE}[K, V], k, k': K, v: V$)

$\text{remove}(\text{put}(t, k', v), k) = \text{put}(\text{remove}(t, k), k', v)$ *iff* $k \neq k'$
(AR1)

$\text{remove}(\text{put}(t, k, v), k) = \text{remove}(t, k)$
iff $\text{has_key}(t, k)$
(AR2)

$\text{remove}(\text{put}(t, k, v), k) = t$
iff $\text{not has_key}(t, k)$
(AR3)

Creators, commands, queries (of T)



Creators: Result type T , arguments do use T

`new` : $\text{TABLE}[K, V]$

`new_with_default_value` : $V \rightarrow \text{TABLE}[K, V]$

Commands: Result type T , arguments use T

`put` : $\text{TABLE}[K, V] \times K \times V \rightarrow \text{TABLE}[K, V]$

`remove` : $\text{TABLE}[K, V] \times K \rightarrow \text{TABLE}[K, V]$

Queries: Result type not T , arguments use T

`item` : $\text{TABLE}[K, V] \times K \rightarrow V$

`has_key` : $\text{TABLE}[K, V] \times K \rightarrow \text{BOOLEAN}$



We call an ADT *consistent (sound)*, if it is not possible to derive a *contradiction*:

$$(A = B) \text{ and } (A \neq B)$$

Sufficient completeness



An ADT is *sufficiently complete* if and only if:

"Every **correct term** where the **outermost function is a query** of the ADT can be reduced, using the axioms of the ADT, into a **term not using any function of the ADT.**"

Sufficient completeness for tables



$\forall t: \text{TABLE}[K, V], k: K, \exists b: \text{BOOLEAN}, v: V$

$\text{has_key}(t, k) = b$

$\text{item}(t, k) = v$ (assuming *item*(*t*, *k*) is correct)

where b and v are terms not using functions of $\text{TABLE}[K, V]$

Proving stuff



Most proves on ADTs use “**structural induction**”:

- We prove the property over all terms produced by **creators** (induction basis: $n = 0$)
- We prove the property over all terms produced by **commands**, assuming the property holds for the **subterms** (induction step: $n \rightarrow n+1$)

Example proof



Prove: any correct term with an outer **remove** is always equivalent to a term with an outer **put** or to the term **new**.

$\forall t: \text{TABLE}[K, V], k:K$

$\exists t': \text{TABLE}[K, V], k':K, v':V:$
remove (t, k) = **put** (t', k', v')

or

remove (t, k) = **new**

Strategy: **Structural induction** over t, proving existence by providing an **example**.

Induction basis ($n=0$)



Show that the theorem holds for any **creator**:

`remove (new, k)`

Induction basis (n=0)



Show that the theorem holds for any **creator**:

remove (**new**, k)

Easy: Not a correct term! Nothing to prove ...

Remember:

“any **correct term** with an outer **remove** is always equivalent to a term with an outer **put** or to the term **new**”

Induction step ($n \rightarrow n+1$)



Show that the theorem holds for any **command** on t (assuming that the theorem holds for t):

assuming for t'' is t or any subterm of t (ind. hypothesis)

$\exists t':k',v':$ **remove** (t'', k'') = **put** (t', k', v') **or** (IH)
remove (t'', k'') = **new**

we have to prove (ind. step)

$\exists t':k',v':$ **remove** (**put** (t, k'', v''), k) = **put** (t', k', v') **or**
remove (**put** (t, k'', v''), k) = **new**

$\exists t':k',v':$ **remove** (**remove** (t, k''), k) = **put** (t', k', v') **or**
remove (**remove** (t, k''), k) = **new**

Induction step for "put"



$\exists t':k',v'$: **remove** (**put** (t, k'', v''), k) = **put** (t', k', v') **or**
remove (**put** (t, k'', v''), k) = **new**

CASE 1: $k \neq k''$

CASE 2: $k = k''$ **and** **has_key** (t, k)

CASE 3: $k = k''$ **and not** **has_key** (t, k)

Induction step for "put" - case 1



$\exists t':k',v':$ **remove** (put (t, k'', v''), k) = **put** (t', k', v') **or**
remove (put (t, k'', v''), k) = **new**

CASE 1: $k \neq k''$

Induction step for "put" - case 1



$\exists t':k',v':$ **remove** (**put** (t, k'', v''), k) = **put** (t', k', v') **or**
remove (**put** (t, k'', v''), k) = **new**

CASE 1: $k \neq k''$

remove (**put** (t, k'', v''), k) = **(AR1)**

put (**remove** (t, k), k'', v'')

Induction step for "put" - case 2



$\exists t':k',v':$ **remove** (**put** (t, k'', v''), k) = **put** (t', k', v') **or**
remove (**put** (t, k'', v''), k) = **new**

CASE 2: k = k'' **and** **has_key** (t, k)

Induction step for "put" - case 2



$\exists t':k',v':$ **remove** (**put** (t, k'', v''), k) = **put** (t', k', v') **or**
remove (**put** (t, k'', v''), k) = **new**

CASE 2: k = k'' **and** **has_key** (t, k)

remove (**put** (t, k'', v''), k) = **remove** (t, k) **(AR2)**

Induction step for "put" - case 2



$\exists t':k',v':$ **remove** (put (t, k'', v''), k) = **put** (t', k', v') **or**
remove (put (t, k'', v''), k) = **new**

CASE 2: k = k'' **and** has_key (t, k)

remove (put (t, k'', v''), k) = **remove** (t, k) (AR2)

So we can reduce the prove goal to:

$\exists t':k',v':$ **remove** (t, k) = **put** (t', k', v') **or**
remove (t, k) = **new**

Proved by (IH)

Induction step for "put" - case 3



$\exists t':k',v'$: **remove** (**put** (t, k'', v''), k) = **put** (t', k', v') **or**
remove (**put** (t, k'', v''), k) = **new**

CASE 3: k = k'' **and not** **has_key** (t, k)

Induction step for "put" - case 3



$\exists t':k',v':$ `remove (put (t, k'', v''), k) = put (t', k', v')` **or**
`remove (put (t, k'', v''), k) = new`

CASE 3: `k = k''` and **not** `has_key (t, k)`

`remove (put (t, k'', v''), k) = t` (AR3)

Induction step for "put" - case 3



$\exists t':k',v':$ `remove (put (t, k'', v''), k) = put (t', k', v')` **or**
`remove (put (t, k'', v''), k) = new`

CASE 3: `k = k''` and not `has_key (t, k)`

`remove (put (t, k'', v''), k) = t` (AR3)

$\exists t':k',v':$ `t = put (t', k', v')` **or**
`t = new`

Should `t` be of the form

`remove (t'', k''')`

we use (IH).

Otherwise we are finished.

Induction step for "remove"



$\exists t':k',v':$ **remove** (**remove** (t, k''), k) = **put** (t', k', v') **or**
remove (**remove** (t, k''), k) = **new**

Induction step for "remove"



$\exists t':k',v':$ **remove** (remove (t, k''), k) = **put** (t', k', v') **or**
remove (remove (t, k''), k) = **new**

We know from (IH)

$\exists t':k',v':$ **remove** (t, k'') = **put** (t', k', v') **or** (IH)
remove (t, k'') = **new**

Induction step for "remove"



$\exists t':k',v':$ **remove** (remove (t, k''), k) = **put** (t', k', v') **or**
remove (remove (t, k''), k) = **new**

We know from (IH)

$\exists t':k',v':$ **remove** (t, k'') = **put** (t', k', v') **or** (IH)
remove (t, k'') = **new**

We have two cases, replacing the *inner* remove:

CASE 1:

$\exists t':k',v':$ **remove** (put (t''', k''', v'''), k) = **put** (t', k', v') **or**
remove (put (t''', k''', v'''), k) = **new**

CASE 2:

$\exists t':k',v':$ **remove** (new, k) = **put** (t', k', v') **or**
remove (new, k) = **new**

Induction step for "remove"



$\exists t':k',v':$ **remove** (remove (t, k''), k) = **put** (t', k', v') **or**
remove (remove (t, k''), k) = **new**

We know from (IH)

$\exists t':k',v':$ **remove** (t, k'') = **put** (t', k', v') **or** (IH)
remove (t, k'') = **new**

We have two cases, replacing the *inner* remove:

CASE 1:

$\exists t':k',v':$ **remove** (put (t''', k''', v'''), k) = **put** (t', k', v') **or**
remove (put (t''', k''', v'''), k) = **new**

CASE 2:

$\exists t':k',v':$ **remove** (new, k) = **put** (t', k', v') **or**
remove (new, k) = **new**

Both proved already !

Summary



- basis of object-orientation

Summary



- basis of object-orientation
- **very important concept** of CS

Summary



- basis of object-orientation
- **very important concept** of CS
- lots of math

Summary



- basis of object-orientation
- **very important concept** of CS
- lots of math
- nice to torture students ;-)
 - have fun with the exercise
 - will most-likely come up in the exam