Functional correctness via refinement

Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

15 Octoner 2014

Outline

1. Abstract Data Types
2. Refinement
3. ADT Transition Systems
4. Phrasing refinement conditions in VCC

ADT type

An *ADT type* is a finite set N of *operation names*.

- Each operation name n in N has an associated *input type* I_n and an *output type* O_n, each of which is simply a set of values.
- We require that there is a special *exceptional value* denoted by e, which belongs to each output type O_n; and that the set of operations N includes a designated *initialization operation* called *init*.

ADT definition

A (deterministic) *ADT* of type N is a structure of the form

$$\mathcal{A} = (Q, U, E, \{op_n\}_{n \in N})$$

where

- Q is the set of states of the ADT,
- $U \in Q$ is an arbitrary state in Q used as an *uninitialized* state,
- $E \in Q$ is an *exceptional* state.
- Each op_n is a *realisation* of the operation n given by $op_n : Q \times I_n \to Q \times O_n$ such that $op_n(E, -) = (E, e)$ and $op_n(p, a) = (q, e) \implies q = E$.
- Further, we require that the *init* operation depends only on its argument and not on the originating state: thus $init(p, a) = init(q, a)$ for each $p, q \in Q \setminus \{E\}$ and $a \in I_{init}$.

ADT type example: Queue

QType

ADT type $QType = \{init, enq, deq\}$ with

$$
\begin{array}{rcl}
I_{init} & = & \{nil\}, \\
O_{init} & = & \{ok, e\}, \\
I_{enq} & = & \mathbb{B}, \\
O_{enq} & = & \{ok, fail, e\}, \\
I_{deq} & = & \{nil\}, \\
O_{deq} & = & \mathbb{B} \cup \{fail, e\}.
\end{array}
$$

Here $\mathbb{B}$ is the set of bit values $\{0, 1\}$, and *nil* is a "dummy" argument for the operations *init* and *deq*.

ADT example: Queue (parameterized by length k) of type $QType$

$QADT_k$

$$QADT_k = (Q, U, E, \{op_n\}_{n \in QType}) \text{ where}$$

$$Q = \{\epsilon\} \cup \bigcup_{i=1}^{k} \mathbb{B}^i \cup \{E\}$$

$$op_{init}(q, a) = \begin{cases} (\epsilon, ok) & \text{if } q \neq E \\ (E, e) & \text{otherwise.} \end{cases}$$

$$op_{enq}(q, a) = \begin{cases} (q \cdot a, ok) & \text{if } q \neq E \text{ and } |q| < k \\ (E, e) & \text{otherwise.} \end{cases}$$

$$op_{deq}(q, a) = \begin{cases} (q', b) & \text{if } q \neq E \text{ and } q = b \cdot q' \\ (E, e) & \text{otherwise.} \end{cases}$$

Language of sequences of operation calls of an ADT

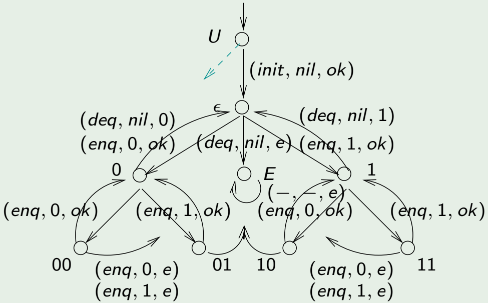
- An ADT $\mathcal{A} = (Q, U, E, \{op_n\}_{n \in N})$ of type N induces a (deterministic) transition system $\mathcal{S}_\mathcal{A} = (Q, \Sigma_N, U, \Delta)$ where
  - $\Sigma_N = \{(n, a, b) \mid n \in N, a \in I_n, b \in O_n\}$ is the set of *operation call* labels corresponding to the ADT type N. The action label (n, a, b) represents a call to operation n with input a that returns the value b.
  - Δ is given by

$$(p, (n, a, b), q) \in \Delta \text{ iff } op_n(p, a) = (q, b).$$

- We define the language of *initialised sequences of operation calls* of $\mathcal{A}$, denoted $L_{init}(\mathcal{A})$, to be $L(\mathcal{S}_\mathcal{A}) \cap ((init, -, -) \cdot \Sigma_N^*)$.
- We say a sequence of operation calls w is *exception-free* if no call in it returns the exceptional value e (i.e. w does not contain a call of the form $(-, -, e)$).

Example: Transition system induced by $QADT_2$

TS induced by $QADT_2$



Refinement between ADT's

Let $\mathcal{A}$ and $\mathcal{B}$ be ADT's of type N. We say $\mathcal{B}$ refines $\mathcal{A}$, written

$$\mathcal{B} \preceq \mathcal{A},$$

iff each exception-free sequence in $L_{init}(\mathcal{A})$ is also in $L_{init}(\mathcal{B})$.

Examples of refinement:

- $QADT_3$ refines $QADT_2$.
- Let $QADT_2'$ be the version of $QADT_2$ where we check for emptiness/fullness of queue and return *fail* instead of e. Then $QADT_2'$ refines $QADT_2$.

Transitivity of refinement

It follows immediately from its definition that refinement is transitive:

Proposition

Let $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ be ADT's of type N, such that $\mathcal{C} \preceq \mathcal{B}$, and $\mathcal{B} \preceq \mathcal{A}$. Then $\mathcal{C} \preceq \mathcal{A}$.

Refinement Condition (RC)

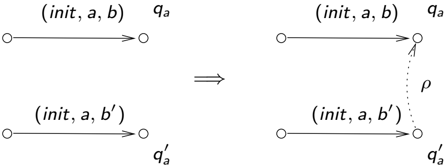
Let $\mathcal{A} = (Q, U, E, \{op_n\}_{n \in N})$ and $\mathcal{A}' = (Q', U', E', \{op_n\}_{n \in N})$ be ADT's of type N. We formulate an *equivalent* condition for $\mathcal{A}'$ to refine $\mathcal{A}$, based on an "abstraction relation" that relates states of $\mathcal{A}'$ to states of $\mathcal{A}$. We say $\mathcal{A}$ and $\mathcal{A}'$ satisfy condition (RC) if there exists a relation $\rho \subseteq Q' \times Q$ such that:

(init) Let $a \in I_{init}$ and let (q_a, b) and (q'_a, b') be the resultant states and outputs after an $init(a)$ operation in $\mathcal{A}$ and $\mathcal{A}'$ respectively, with $b \neq e$. Then we require that $b = b'$ and $(q'_a, q_a) \in \rho$.

(sim) For each $n \in N$, $a \in I_n$, $b \in O_n$, and $p' \in Q'$, with $(p', p) \in \rho$, whenever $p \xrightarrow{(n,a,b)} q$ with $b \neq e$, then there exists $q' \in Q'$ such that $p' \xrightarrow{(n,a,b)} q'$ with $(q', q) \in \rho$.

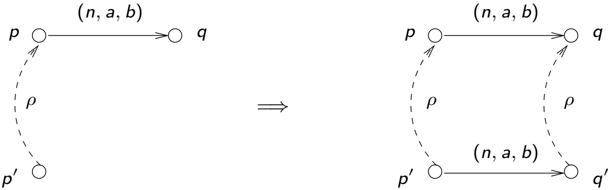
Illustrating condition (RC)

(RC-init):



and $b = b'$

(RC-sim):



Condition (RC) is necessary and sufficient for refinement

Theorem

Let $\mathcal{A}$ and $\mathcal{A}'$ be two ADT's of type N. Then $\mathcal{A}' \preceq \mathcal{A}$ iff they satisfy condition (RC).

Condition (RC) is necessary and sufficient for refinement

Theorem

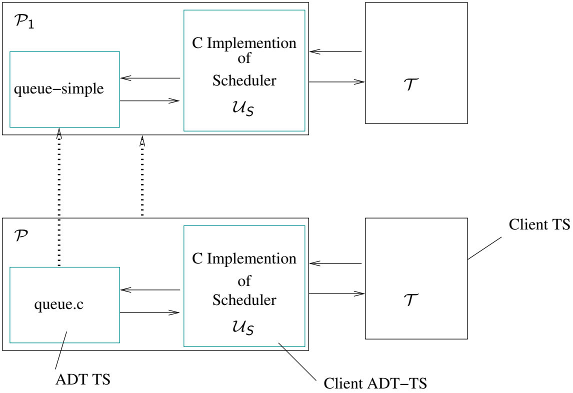
Let $\mathcal{A}$ and $\mathcal{A}'$ be two ADT's of type N. Then $\mathcal{A}' \preceq \mathcal{A}$ iff they satisfy condition (RC).

Exercise

Find an abstraction relation ρ for which $QADT_2$ and $QADT_3$ satisfy condition (RC).

Why ADT Transition Systems

- To reason about imperative implementations of ADT's (read transition-system based implementions)
- To do so compositionally.



A C implementation of a queue

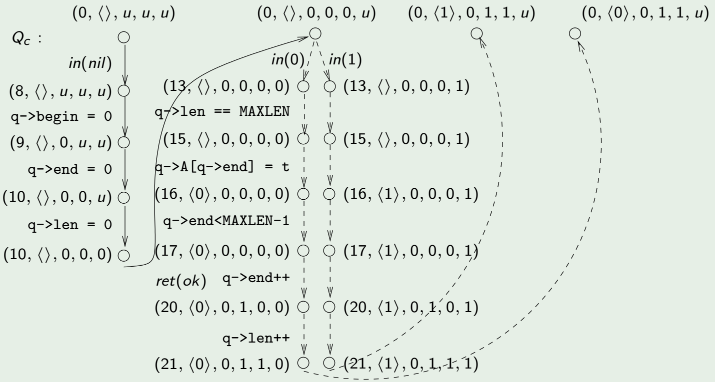
```
int A[MAXLEN];
unsigned beg, end, len;

void init() {
  beg = 0;
  end = 0;
  len = 0;
}

int deq() { ...  }
void enq(int t) {
  if (len == MAXLEN)
    assert(0); /* exception */
  A[end] = t;
  if (end < MAXLEN-1)
    end++;
  else
    end = 0;
  len++;
}
```

Example: ADT Transition System induced by queue.c

Part of the ADT TS induced by queue.c, showing init and enq opns



ADT induced by an ADT TS

An ADT transition system like $\mathcal{S}$ above induces an ADT $\mathcal{A}_{\mathcal{S}}$ of type N given by $\mathcal{A}_{\mathcal{S}} = (Q_c \cup \{E\}, U, E, \{op_n\}_{n \in N})$ where for each $n \in N$, $p \in Q_c \cup \{E\}$, and $a \in I_n$, we have:

$$op_n(p, a) = \begin{cases} (q, b) & \text{if there exists a path of the form} \\ & p \xrightarrow{in(a)} r_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{k-1}} r_k \xrightarrow{ret(b)} q \text{ in } \mathcal{S} \\ (E, e) & \text{otherwise.} \end{cases}$$

We say that an ADT TS $\mathcal{S}'$ refines another ADT TS $\mathcal{S}$ iff $\mathcal{A}_{\mathcal{S}'}$ refines $\mathcal{A}_{\mathcal{S}}$.

Substitutability Claim

We claim that refinement is "substitutive" and gives us a compositional way of reasoning about ADT implementations:

Theorem

Let $\mathcal{U}$ be an M-client ADT transition system of type N, and $\mathcal{B}$ and $\mathcal{C}$ be ADT's of type M such that $\mathcal{C} \preceq \mathcal{B}$. Then we have $\mathcal{A}_{\mathcal{U}[\mathcal{C}]} \preceq \mathcal{A}_{\mathcal{U}[\mathcal{B}]}$.

Refinement Conditions in VCC: Using a ghost model

(init-a) $\mathtt{func}_{init}^{\mathcal{M},\mathcal{P}}$ terminates on all joint state-input pairs satisfying $pre_{init}^{\mathcal{M}}$.

(init-b) $\mathtt{func}_{init}^{\mathcal{M},\mathcal{P}}$ $(\mathtt{X}_{init}\ \mathtt{x})$

$\mathtt{\_(requires}\ pre_{init}^{\mathcal{M}}\mathtt{)}$

$\mathtt{\_(ensures}\ inv_{\rho} \wedge y_{init}^{\mathcal{M}} = y_{init}^{\mathcal{P}}\mathtt{)}$ {

  // body of $\mathtt{func}_{init}^{\mathcal{M}}$

  // body of $\mathtt{func}_{init}^{\mathcal{P}}$

}

(sim-a) For each operation n, $\mathtt{func}_{n}^{\mathcal{M},\mathcal{P}}$ must terminate on all state-input pairs satisfying $pre_{n}^{\mathcal{M}} \wedge inv_{\rho}$.

(sim-b) For each operation n:

$\mathtt{func}_{n}^{\mathcal{M},\mathcal{P}}$ $(\mathtt{X}_{n}\ \mathtt{x})$

$\mathtt{\_(requires}\ pre_{n}^{\mathcal{M}} \wedge inv_{\rho}\mathtt{)}$

$\mathtt{\_(ensures}\ inv_{\rho} \wedge y_{n}^{\mathcal{M}} = y_{n}^{\mathcal{P}}\mathtt{)}$ {

  // body of $\mathtt{func}_{n}^{\mathcal{M}}$

  // body of $\mathtt{func}_{n}^{\mathcal{P}}$

}

Some examples: store with get/put

```
struct store {
   int val1;
   int val2;
   int flag;
} s;

void init() {
   s.flag = 1;
   s.val1 = 0;
}

void set(int x) {
   if (s.flag)
       s.val2 = x;
   else
       s.val1 = x;
   s.flag = (1 - s.flag);
}
```

```
int get(void) {
    if (s.flag)
        return s.val1;
    else
        return s.val2;
 }
```

Overall strategy for refinement