Certification of Data Structures

Kurt Mehlhorn

MPI für Informatik
Saarbrücken
Germany

Data Structures or Reactive Programs

MAX-PLANCK-GESELLSCHAFT

- reactive programs run forever, receive stimuli and respond to them. Algorithms community calls them data structures. Data structures implement abstract data types

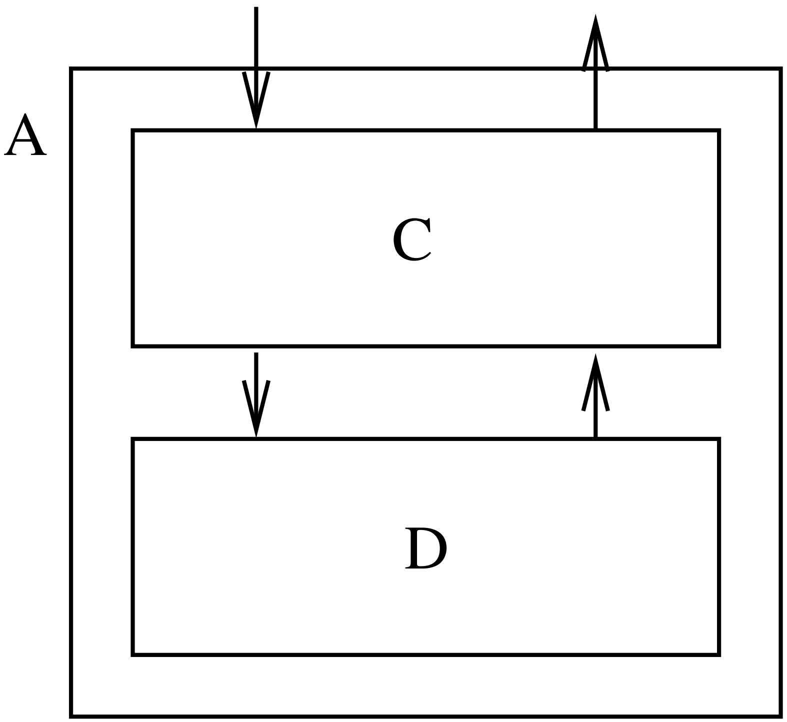
- an abstract data type has an (usually infinite) set S of states; input x leads to a change of state and maybe also an ouput in Y

$$\delta : S \times X \mapsto S \times Y \cup \{\varepsilon\}$$

- query: no change of state   update: change of state

- an implementation also has a set S' of states and a transition function δ'
- implementation is correct (Hoare) if there is a function $rep : S' \mapsto S$ s.t. for all x, s', y, t' with $\delta'(s',x) = (t',y)$ we have $\delta(rep(s'),x) = (rep(t'),y)$

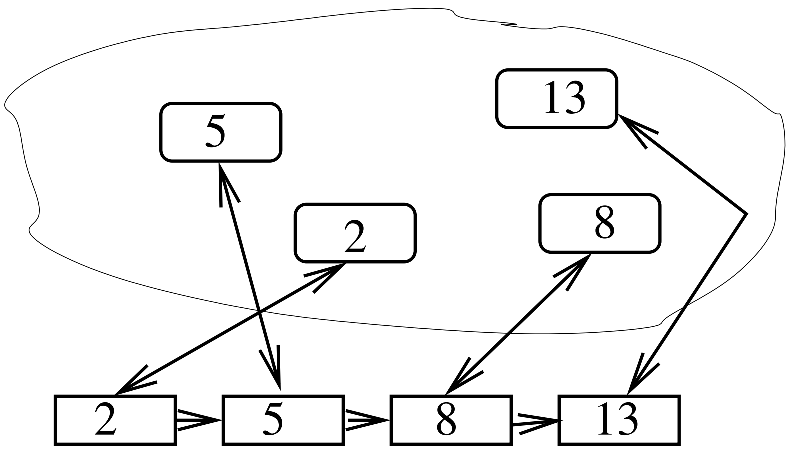
Monitoring Data Stuctures



- D is the implementation of some abstract data type A'
- C monitors its behavior.
- Any input from the environment is passed to C which then forwards it, maybe in modified form, to D. D reacts to it, C inspects the reaction of D and returns an answer to the environment.
- If D is correct, the combination of C and D realizes the abstract data type A, if D is incorrect, C catches the error.

- immediately (fail-stop) or ultimately
- $A' = A$ or A' more powerful than A.
- want C to be less complex than D (simpler, faster)

Ordered Dictionaries



- The dictionary problem for a universe U and a set I of informations asks to maintain a set S of pairs $(x,i) \in U \times I$ with pairwise-distinct keys (= first elements) under operations $insert(x,i)$, $delete(h)$, and $find(x)$. Here, h is a handle to a pair in the dictionary. $insert(x,i)$ returns a handle.
- $locate(x)$ returns a handle to a pair $(y, \ ) \in S$ with $y \leq x$ and y maximal.

- C maintains a sorted list, one for each item in S. Information is pointer to the corresponding pair in the dictionary implementation.
- C requires constant time per operation
- without locate, C requires logarithmic time per operation

Monitoring Priority Queues I

a PQ maintains a set S (of real numbers) under the operations insert and delete_min

$$insert(5),\quad insert(2),\quad insert(4),\quad delete\_min,\quad insert(7),\quad delete\_min$$

Monitoring Priority Queues I

a priority queue maintains a set S (of real numbers) under the operations insert and delete_min

$insert(5)$, $insert(2)$, $insert(4)$, $delete\_min$, $insert(7)$, $delete\_min$

$insert(5)$,	$insert(2)$,	$insert(4)$,	$delete\_min$,	$insert(7)$,	$delete\_min$
			must return 2		must return 4

Monitoring Priority Queues I

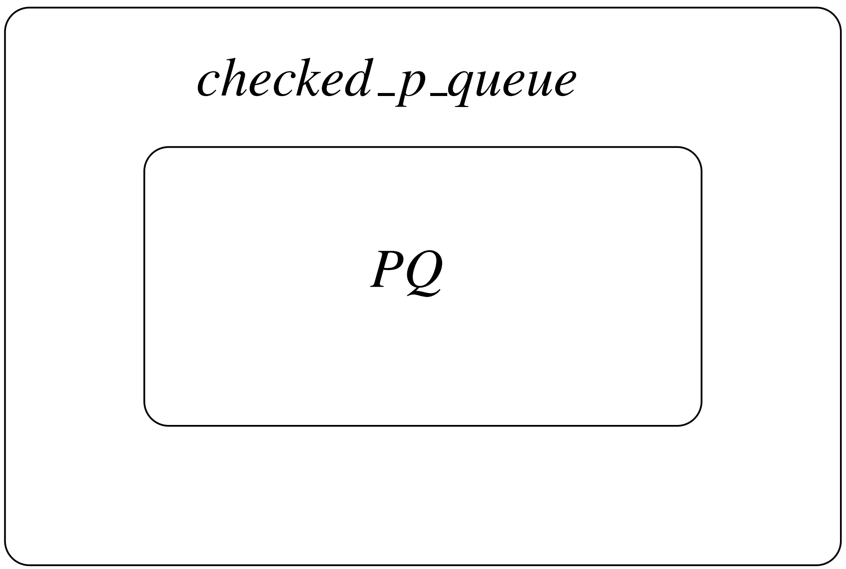
a PQ maintains a set S (of real numbers) under the operations insert and delete_min

$insert(5)$,	$insert(2)$,	$insert(4)$,	$delete\_min$,	$insert(7)$,	$delete\_min$
			must return 2		must return 4
			returns 2		return 5

Monitoring Priority Queues I

a PQ maintains a set S (of real numbers) under the operations insert and delete_min

$insert(5)$,	$insert(2)$,	$insert(4)$,	$delete\_min$,	$insert(7)$,	$delete\_min$
			must return 2		must return 4
			returns 2		return 5



A checker wraps around any priority queue PQ and monitors its behavior.

- It offers the functionality of a priority queue
- It complains if PQ does not behave like a priority queue.
  - immediately
  - ultimately

Monitoring Priority Queues II

Fact: Priority queue implementations with logarithmic running time per operation exist.

Fact:

- There is a checker with additional constant amortized running time per operation.
  It catches errors ultimately, namely with linear delay
- Immediate error catching requires $\Omega(\log n)$ additional time per operation.

Finkler/Mehlhorn, SODA 99

Monitoring Priority Queues: The Upper Bound

- Checker maintains elements in queue in linear list ordered by time of insertion
- deletemin:
  - check, whether the element returned by the oracle, has required minimal value
  - if so, lift the step containing it and all steps to the right to the new minimal value
- insert: extend linear list by one element
- efficient implementation: union-find