Correct tout seul, sûr à plusieurs

Clément Allain, Gabriel Scherer

INRIA (Cambium), INRIA (Picube)

30 janvier 2024
This talk

The story of how we got Dynarray in the OCaml standard library.
This talk

The story of how we got Dynarray in the OCaml standard library.

... and the horrors that lie beneath ...
This talk

The story of how we got Dynarray in the OCaml standard library.

... and the horrors that lie beneath ...

with Coq proofs!
Dynarray : what?

An array...

val init : int -> (int -> 'a) -> 'a t

val get : 'a t -> int -> 'a

val set : 'a t -> int -> 'a -> unit

val length : 'a t -> int
Dynarray : what?

An array...

val init : int -> (int -> 'a) -> 'a t

val get : 'a t -> int -> 'a
val set : 'a t -> int -> 'a -> unit

val length : 'a t -> int

that is also a stack (Daniel Bünzli):

val create : unit -> 'a t
val add_last : 'a t -> 'a -> unit
val pop_last_opt : 'a t -> 'a option
Dynarray: why?

- You want to build an array by accumulating elements, but you don’t know the size in advance. (Note: Array.of_list may also work very well.)
- You want a stack or bag, but also indices and random access.

Classic examples:

- Priority queues stored in an array (textbook algorithm).
  Stdlib priority queues (@backtracking, January 2024)
  https://github.com/ocaml/ocaml/pull/12871
- The journal of a journalled data structure.
- The trail of a SAT/SMT solver.
- Clause sets in an automated prover.
Dynarray : how?

Implementation (’a slot is a secret for now):

```ocaml
type ’a t = {
    mutable data : ’a slot array;
    mutable len : int;
}
```

![Diagram](image)

Capacity (backing array length). Space control (Simon Cruanes):

```ocaml```

```ocaml
val capacity : ’a t -> int
val ensure_capacity : ’a t -> int -> unit
val fit_capacity : ’a t -> unit
```
Story time (1)

Once upon a time, a brave, brave contributor
Story time (1)
Once upon a time, a brave, brave contributor

(Simon Cruanes)
Story time (1)

Once upon a time, a brave, brave contributor

(Simon Cruanes)

wanted to improve the OCaml standard library by adding a Dynarray module from his containers library. Many had tried before him...

- ...

- https://discuss.ocaml.org/t/adding-dynamic-arrays-vectors-to-stdlib/4697/38
- https://github.com/ocaml/ocaml/pull/9122
Story time

He held a secret meeting with two gate keepers of the stdlib
Story time

He held a secret meeting with two gate keepers of the stdlib

They brainstormed an API, and a PR was born.

“add ‘Dynarray‘ to the stdlib” (@c-cube, September 2022)
https://github.com/ocaml/ocaml/pull/11563
Horror 1: empty value

What value should we store in the empty space?

A user-provided default value: inconvenient API.

The last user-provided value: space leak.

Obj.magic(): ew.

None: ew. ('a option array)
let[@inline] get v i =
    if i < 0 || i >= v.len then
        invalid_arg "CCVector.get";
        Array.unsafe_get v.data i

What if another domain races on v.len?

unsafe_get : segfault (out of backing array)

Obj.magic () : segfault (out of user space)
Story, continued

After endless nights fighting the zombie hordes of `Obj.magic()`, the PR went into an eternal sleep.
Story, continued

After endless nights fighting the zombie hordes of Obj.magic (), the PR went into an eternal sleep.

Until:
“Dynarrays, boxed” (@gasche, January 2023)
https://github.com/ocaml/ocaml/pull/11882

type 'a slot =
| Empty
| Elem of { mutable v : 'a }

Reassuring benchmarks.
Problem solved?
Horror 3 : iterator invalidation

val iter : ('a -> unit) -> 'a t -> unit

What happens if elements are added or removed during iter?

1. something reasonable (but slower)?
2. weak memory model?
3. invalid, maybe an error?
4. invalid, always an error?
Story, end

Many more months of feedback, changes, decisions.

Clément Allain reviewed the code for correctness.

Merged in OCaml 5.2! (to be released soon)
Take away

```ocaml
let[@inline] get v i =
  if i < 0 || i >= v.len then
    invalid_arg "CCVector.get";
  Array.unsafe_get v.data i
```

Public announcements:

Library code must remain memory-safe for all uses, including incorrect concurrent code.
Take away

let[@inline] get v i =
  if i < 0 || i >= v.len then
    invalid_arg "CCVector.get";
  Array.unsafe_get v.data i

Public announcements :

  Library code must remain memory-safe for all uses, including incorrect concurrent code.

Consequence :
Some unsafe code that was perfectly fine with OCaml 4 is now unsound with OCaml 5

Time to review all your unsafe_{get,set} calls.
How do we reason about this?
Strong invariant for functional correctness

\[
\begin{align*}
t & \rightarrow \begin{array}{c|c}
len & data \\
\end{array} \\
\end{align*}
\]

\[
\begin{array}{cccccc}
\text{Elem} & \cdots & \text{Elem} & \text{Emp} & \cdots & \text{Emp} \\
\end{array}
\]

\[
\begin{array}{c}
v_0 \\

n_{len-1}
\end{array}
\]
Weak invariant for memory safety

\[
t : \tau \ t
\]

\[
t \rightarrow len \ data \rightarrow \tau \ slot \ \cdots \ \tau \ slot
\]

\[
0 \leq len
\]

\[
slot : \tau \ slot
\]

\[
slot = \text{Emp} \quad \mid \quad slot \rightarrow \tau
\]
A method\(^1\) to reason about sequential/concurrent algorithms using unsafe features in OCAML 5

<table>
<thead>
<tr>
<th>Functional correctness</th>
<th>Memory safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each function respects its specification.</td>
<td>Each function inhabits its semantic type.</td>
</tr>
<tr>
<td>Strong invariant</td>
<td>Weak invariant</td>
</tr>
</tbody>
</table>

---

1. Thanks to Armaël Guéneau for stating clearly the dichotomy.
Reviewing Dynarray

- Dynarray review
- Separation logic (IRIS)
- Coq

= OCAMLBELT
= semantic typing implying memory safety

≈ RUSTBELT
Formalization in the IRIS separation logic (mechanized in Coq)

val create : unit -> 'a t
val make : int -> 'a -> 'a t
val init : int -> (int -> 'a) -> 'a t
val length : 'a t -> int
val get : 'a t -> int -> 'a
val set : 'a t -> int -> 'a -> unit
val add_last : 'a t -> 'a -> unit
val pop_last : 'a t -> 'a
val ensure_capacity : 'a t -> int -> unit
val ensure_extra_capacity : 'a t -> int -> unit
val fit_capacity : 'a t -> unit
val reset : 'a t -> unit
Strong invariant for functional correctness... in \textsc{Iris}

Definition dynarray_model t vs : iProp Σ :=
\[ \exists \ l \ \text{data} \ \text{slots} \ \text{extra}, \]
\[ [t = \#l] \ * \]
\[ l.[\text{len}] \mapsto \#(\text{length vs}) \ * \]
\[ l.[\text{data}] \mapsto \text{data} \ * \]
array_model data (slots ++ replicate extra &&None) *
[* list] slot; v ∈ slots; vs, slot_model slot v.

Lemma dynarray_pop_last_spec t vs v :
\{\{ dynarray_model t (vs ++ \{v\}) \}\}
\text{dynarray_pop_last} t
\{\{ \text{RET v}; \ dynarray_model t vs \}\}. 
Weak invariant for memory safety... in IRIS

Definition dynarray_type $\tau$ ‘{iType _ $\tau$} t : iProp $\Sigma :=$
\[ \exists \; l, \]
\[ t = \#l \]
\[ \text{inv nroot (}
\exists \; \text{l.}[\text{len}] \rightarrow \#\text{len} \]
\[ \text{l.}[\text{data}] \rightarrow \text{data} \]
\[ \text{array_type (slot_type } \tau) \text{ cap data} \]
).

Lemma dynarray_pop_last_type $\tau$ t :
{{{ dynarray_type $\tau$ t }}}{{
 dynarray_pop_last t
{{{ v, RET v; $\tau$ v }}}}.

Definition dynarray_pop_last : val :=
λ: "t",
    let: "len" := dynarray_len "t" in
    let: "arr" := dynarray_data "t" in
    assume ("len" <= array_length "arr") ;;
    assume (#0 < "len") ;;
    let: "last" := "len" - #1 in
    match: array_unsafe_get "arr" "last" with
    | None =>
        diverge #()
    | Some "ref" =>
        array_unsafe_set "arr" "last" &None ;;;
        dynarray_set_size "t" "last" ;;
        !"ref"
    end.
What the mechanized \textbf{IRIS} proofs look like:

Proof.
\begin{verbatim}
  iIntros "\%Φ #Htype Φ".
  wp_rec.
  wp_apply (dynarray_len_type with "Htype") as "\%sz _".
  wp_smart_apply (dynarray_data_type with "Htype") as "\%cap %data #Hdata_type".
  wp_smart_apply (array_size_type with "Hdata_type") as "_".
  wp_smart_apply assume_spec' as "\%Hcap".
  wp_smart_apply assume_spec' as "\%Hsz".
  wp_smart_apply (array_unsafe_get_type with "Hdata_type") as "\%slot #Hslot".
  \{ lia. \}
  wp_apply (opt_type_match with "Hslot"). iSplit.
  - wp_apply diverge_spec.
  - iIntros "\%r #Hr /=".
    wp_smart_apply (array_unsafe_set_type with "[Hdata_type]") as "_".
    \{ lia. \}
    \{ iSteps. \}
    wp_smart_apply (dynarray_set_size_type with "Htype") as "_".
    \{ lia. \}
    wp_smart_apply (reference_get_type with "Hr"). iSteps.
\end{verbatim}

Qed.
Thanks!

Questions?
let pop_last a =
  let {data = arr; len = length} = a in
  check_valid_length length arr;
  (* We know [length <= capacity a]. *)
  if length = 0 then raise Not_found;
  let last = length - 1 in
  (* We know [length > 0] so [last >= 0]. *)
  match Array.unsafe_get arr last with
     | Empty ->
       Error.missing_element ~i:last ~length
     | Elem s ->
       Array.unsafe_set arr last Empty;
       a.length <- last;
       s.v