Program Extraction for Mutable Arrays

Kazuhiko Sakaguchi (University of Tsukuba)

We provide a Coq library [1] for mutable arrays to improve the efficiency of extracted programs.

Representation of Arrays in Coq

Arrays are isomorphic to functions with finite domains, in the theoretical way. We use `fintype` and `finfun` library of the **Mathematical Components (MathComp)** [3] to represent arrays with arbitrary finite index sets. In original MathComp library, `finType`s are defined as types equipped with a finite enumeration. We modified the MathComp library [2] to recharacterize `finType`s by a bijection to a finite ordinal $'\mathrm{I}_n = \{0, \ldots, n-1\}$, because of efficiency of computation.

Array State Monad

We define a specialized state monad called the **array state monad** to represent computations involving mutable arrays:

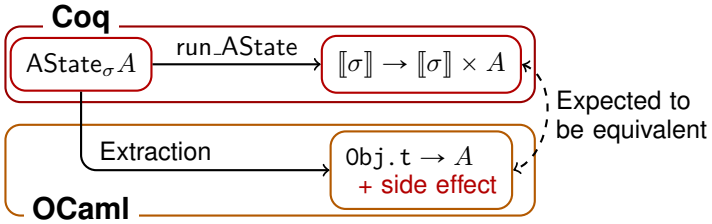
$\mathsf{AState}_\sigma A$ → Return type

List of `(finType * Type)`, and called the *signature*

The signature $\sigma = (I_1, T_1), \ldots, (I_n, T_n)$ indicates that the states are $I_1, \ldots, I_n$ indexed arrays of types $T_1, \ldots, T_n$ respectively.

constructors:

$$\begin{aligned}
&\mathsf{return}(x) : \mathsf{AState}_\sigma A && (x : A),\\
&a \gg\!= f : \mathsf{AState}_\sigma B && (a : \mathsf{AState}_\sigma A, f : A \to \mathsf{AState}_\sigma B),\\
&\mathsf{lift}(a) : \mathsf{AState}_{(I,T)::\sigma} A && (a : \mathsf{AState}_\sigma A),\\
&\mathsf{get}(i) : \mathsf{AState}_{(I,T)::\sigma} T && (i : I),\\
&\mathsf{set}(i, x) : \mathsf{AState}_{(I,T)::\sigma} \mathsf{unit} && (i : I, x : T).
\end{aligned}$$



Improvement of the Extraction Plugin

We improved the extraction of `match` expressions by η-expansions with distribution of arguments to each branch. The combination of this conversion and inlining by the Flambda optimizer reduces the cost of function calls, especially in array state monad programming. For example:

Input (* `if` expressions are a special case of `match` expressions)

$\mathsf{get}(i) \gg\!= \mathsf{fun}\ x \Rightarrow \mathsf{if}\ \mathsf{odd}(x)\ \mathsf{then}\ \mathsf{set}(i, x+1)\ \mathsf{else}\ \mathsf{set}(i, x+2)$

Official version of extraction

```
(fun f g s -> let r = f s in g r s)
((fun i s -> s.(i)) i)
(fun x -> if odd x
 then (fun i x s -> s.(i) <- x)
     i (x + 1)
 else (fun i x s -> s.(i) <- x)
     i (x + 2))
```

Inlining

```
fun s ->
let r = s.(i) in
(if odd r
 then (fun s -> s.(i) <- r + 1)
 else (fun s -> s.(i) <- r + 2)) s
```

Improved extraction

```
(fun f g s -> let r = f s in g r s)
((fun i s -> s.(i)) i)
(fun x s -> if odd x
 then (fun i x s -> s.(i) <- x)
     i (x + 1) s
 else (fun i x s -> s.(i) <- x)
     i (x + 2) s)
```

Inlining

```
fun s ->
let r = s.(i) in
if odd r
then s.(i) <- r + 1
else s.(i) <- r + 2
```

Benchmarks

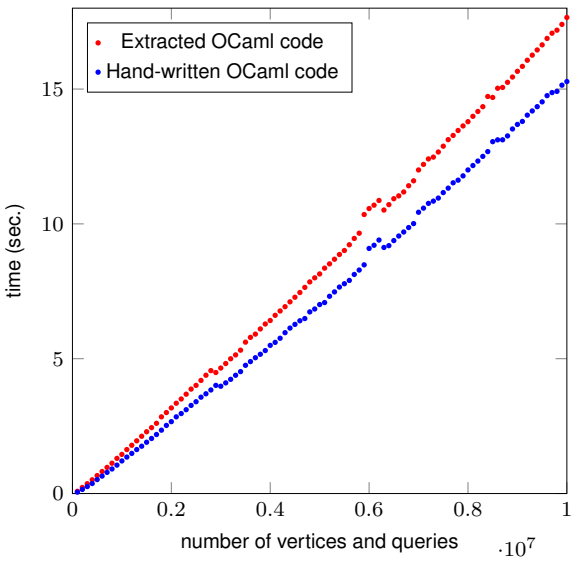


Figure 1: The benchmark results of the Union–Find data structure with path compression and weighted union: for each integer n $(0 < n \leqslant 10^7, n = 0 \mod 10^5)$, this benchmark initializes a Union–Find tree with n vertices and performs n-times random unions and n-times random finds in order.

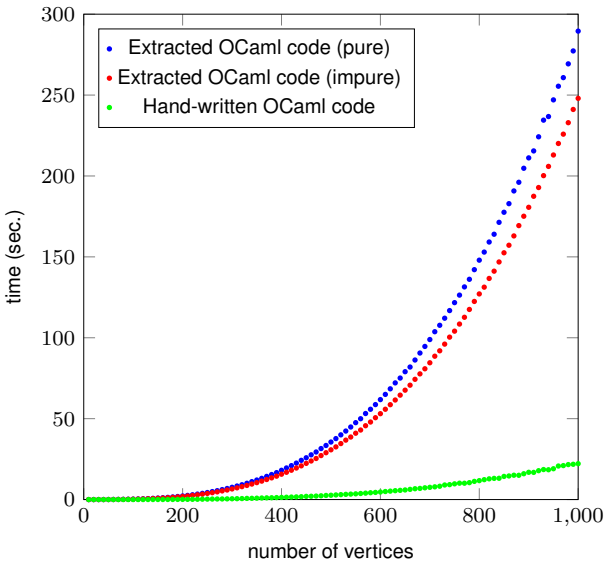


Figure 2: The benchmark results of the Floyd–Warshall shortest path algorithm: for each integer n $(0 < n \leqslant 1000, n = 0 \mod 10)$, this benchmark generates a weighted directed graph with n vertices randomly, and computes the lengths of the shortest paths between all pairs of the vertices.

References

[1] Kazuhiko Sakaguchi. *Extraction of Efficient Programs Which Handle Mutable Arrays.* URL: https://github.com/pi8027/efficient-finfun.

[2] Kazuhiko Sakaguchi and Yukiyoshi Kameyama. "Efficient Finite-Domain Function Library for the Coq Proof Assistant". Japanese. In: *IPSJ Transactions on Programming* 10.1 (2016), pp. 14–28. URL: http://logic.cs.tsukuba.ac.jp/~sakaguchi/papers/ipsj-pro-2016-1-7.pdf.

[3] The Mathematical Components project. *The Mathematical Components repository.* URL: https://github.com/math-comp/math-comp.