Verified multi-word compare-and-set and software transactional memory for OCaml 5

ANONYMOUS AUTHOR(S)

1 INTRODUCTION
2 OVERVIEW OF THE LIBRARY
3 IMPLEMENTATION OF THE LIBRARY
4 MULTI-WORD COMPARE-AND-SET ALGORITHM
5 BENCHMARKS
6 VERIFICATION OF MULTI-WORD COMPARE-AND-SET ALGORITHM
7 RELATED WORK
8 CONCLUSION

```
type 'a loc =
  { atomic state: 'a state;
    id: int;
  }
and 'a state =
  { casn: 'a casn;
    mutable before: 'a;
    mutable after: 'a;
  }
and 'a cas =
  { loc: 'a loc;
    state: 'a state;
  }
and 'a casn =
  { atomic status: 'a status;
    proph: (ghost_id * bool) proph;
  }
and 'a status =
  | Undetermined of 'a cas list
  | Before
  | After
```

Fig. 1. Type definitions for implementing multi-word compare-and-set

```
let status_to_bool = function
  | Undetermined _ -> assert false
  | Before -> false
  | After -> true
let finish gid casn status =
  match casn.status with
  | Before -> false
  | After -> true
  | Undetermined _ as old_status ->
      resolve (
        Atomic.Loc.compare_and_set [%atomic.loc casn.status] old_status status
      ) casn.proph (gid, status_to_bool status) |> ignore ;
      casn.status == After

let rec determine_as casn cass =
  let gid = ghost_id in
  match cass with
  | [] ->
      finish gid casn After
  | cas :: cass' ->
      let { loc; state } = cas in
      let state' = loc.state in
      if state == state' then
        determine_as casn cass'
      else
        let v = get_as state' in
        if get_as state' != state.before then
          finish gid casn Before
        else
          match casn.status with
          | Before -> false
          | After -> true
          | Undetermined _ ->
              if Atomic.Loc.compare_and_set [%atomic.loc loc.state] state' state
              then determine_as casn cass'
              else determine_as casn cass
and get_as state =
  if determine state.casn then state.after else state.before
and determine casn =
  match casn.status with
  | Before -> false
  | After -> true
  | Undetermined cass -> determine_as casn cass
```

Fig. 2. Implementation of multi-word compare-and-set (1)

```
let make v id =
  let _gid = ghost_id in
  let casn = { status= After; proph= proph } in
  let state = { casn; before= v; after= v } in
  Atomic.make { state; id }

let get loc =
  get_as loc.state

let cas cass =
  let casn = { status= After; proph= proph } in
  let cass =
    Lst.map cass (fun (loc, before, after) ->
      let state = { casn; before; after } in
      { loc; state }
    )
  in
  casn.status <- Undetermined cass ;
  determine_aux casn cass
```

Fig. 3. Implementation of multi-word compare-and-set (2)

REFERENCES