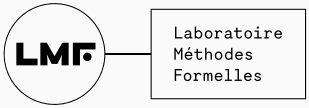
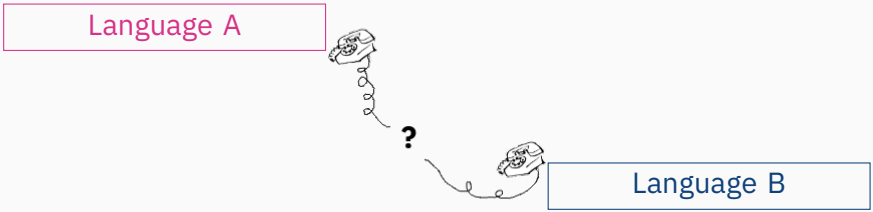
Formalisation of interoperability between C and OCaml

Gurvan DEBAUSSART Valeran MAYTIE

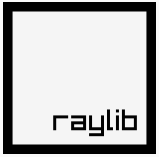


Language interoperability



Foreign function interface

Use case: Using a library made for another language





Number of bindings	75	13
Original language	C	C

OCaml FFI

xavierleroy/
cryptokit



A library of cryptographic primitives (ciphers, hashes, etc) for OCaml

12 Contributors · 4 Issues · 94 Stars · 23 Forks

dbuenzli/**tsdl**

Thin bindings to SDL for OCaml

Languages

OCaml 92.5% · C 7.5%

```
val execv : string -> string array -> 'a
val execvp : string -> string array -> 'a

val fork : unit -> int

val getpid : unit -> int
val getppid : unit -> int

...
```

bindings in stdlib

Example: add one

```
external add_one : int -> int =
  "caml_add_one"
let two = add_one 1
```

```
int caml_add_one(int i) {
  return 1 + i;
}
```

Example: add one

```
external add_one : int -> int =
  "caml_add_one"
let two = add_one 1
```

```
int caml_add_one(int i) {
  return 1 + i;
}
```



Integer



Pointer

Example: add one

```
external add_one : int -> int =
  "caml_add_one"
let two = add_one 1
```

```
value caml_add_one(value i) {
  return Val_int(1 + Int_val(i));
}
```



Integer



Pointer

Example: increment

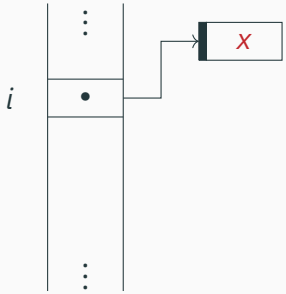
```
external incr : int ref -> unit =
  "caml_incr"
```

```
value caml_incr(value i) {
  int c = 1 + Int_val(Field(i, 0));
  Store_field(i, 0, Val_int(c));
  return Val_unit;
}
```

Example: increment

```
external incr : int ref -> unit =
  "caml_incr"
```

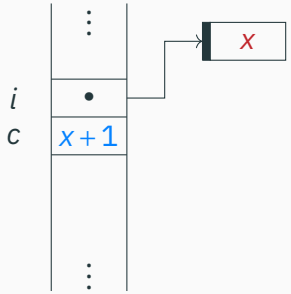
```
value caml_incr(value i) {                    ⇐
  int c = 1 + Int_val(Field(i, 0));
  Store_field(i, 0, Val_int(c));
  return Val_unit;
}
```



Example: increment

```
external incr : int ref -> unit =
  "caml_incr"
```

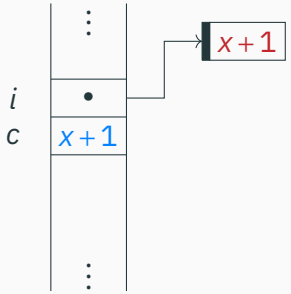
```
value caml_incr(value i) {
  int c = 1 + Int_val(Field(i, 0));     ⇐
  Store_field(i, 0, Val_int(c));
  return Val_unit;
}
```



Example: increment

```
external incr : int ref -> unit =
  "caml_incr"
```

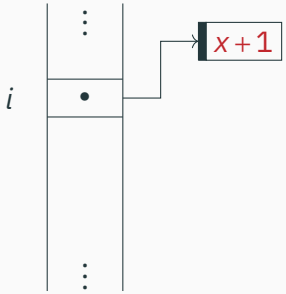
```
value caml_incr(value i) {
  int c = 1 + Int_val(Field(i, 0));
  Store_field(i, 0, Val_int(c));        ⇐
  return Val_unit;
}
```



Example: increment

```
external incr : int ref -> unit =
  "caml_incr"
```

```
value caml_incr(value i) {
  int c = 1 + Int_val(Field(i, 0));
  Store_field(i, 0, Val_int(c));
  return Val_unit;                    ⇐
}
```



Example: swap pair

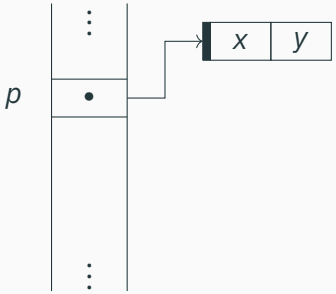
```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

```
value caml_swap_pair(value p) {
  value r = caml_alloc(2, 0);
  Store_field(r, 0, Field(p, 1));
  Store_field(r, 1, Field(p, 0));
  return r;
}
```

Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

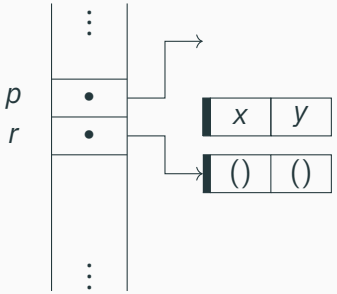
```
value caml_swap_pair(value p) {    ⇐
  value r = caml_alloc(2, 0);
  Store_field(r, 0, Field(p, 1));
  Store_field(r, 1, Field(p, 0));
  return r;
}
```



Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

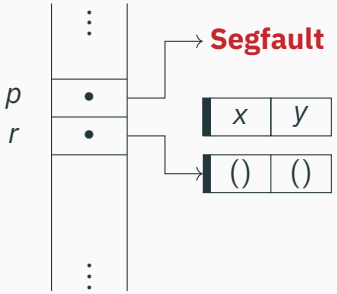
```
value caml_swap_pair(value p) {
  value r = caml_alloc(2, 0);      ⇐
  Store_field(r, 0, Field(p, 1));
  Store_field(r, 1, Field(p, 0));
  return r;
}
```



Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

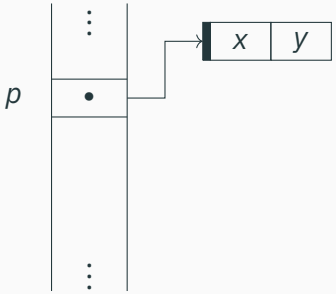
```
value caml_swap_pair(value p) {
  value r = caml_alloc(2, 0);
  Store_field(r, 0, Field(p, 1)); ⇐
  Store_field(r, 1, Field(p, 0));
  return r;
}
```



Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

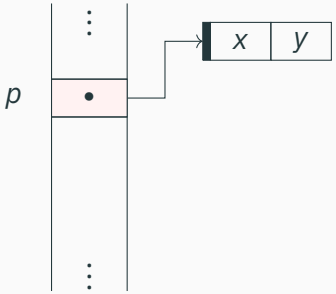
```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  Store_field(r, 0, Field(p, 1));
  Store_field(r, 1, Field(p, 0));
  CAMLreturn(r);
}
```



Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

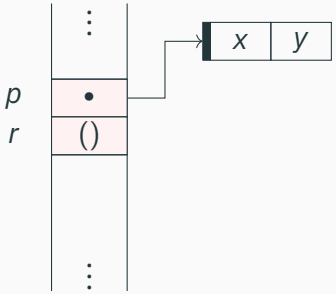
```
value caml_swap_pair(value p) {
  CAMLparam1(p);                    ⇐
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  Store_field(r, 0, Field(p, 1));
  Store_field(r, 1, Field(p, 0));
  CAMLreturn(r);
}
```



Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

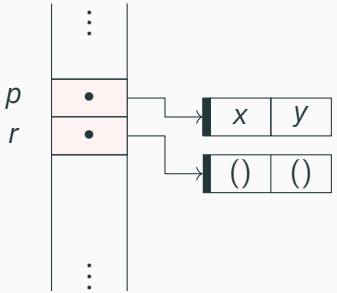
```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);                    ⇐
  r = caml_alloc(2, 0);
  Store_field(r, 0, Field(p, 1));
  Store_field(r, 1, Field(p, 0));
  CAMLreturn(r);
}
```



Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

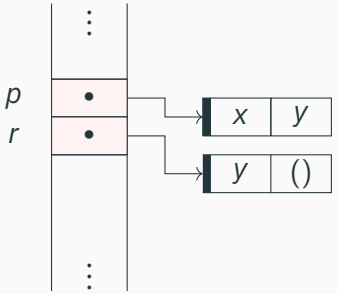
```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);                ⇐
  Store_field(r, 0, Field(p, 1));
  Store_field(r, 1, Field(p, 0));
  CAMLreturn(r);
}
```



Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

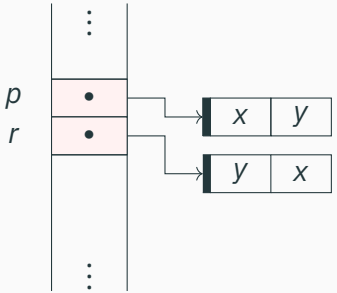
```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  Store_field(r, 0, Field(p, 1)); ⇐
  Store_field(r, 1, Field(p, 0));
  CAMLreturn(r);
}
```



Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

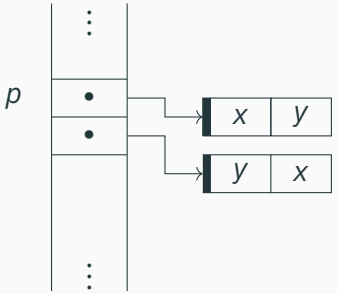
```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  Store_field(r, 0, Field(p, 1));
  Store_field(r, 1, Field(p, 0)); ⇐
  CAMLreturn(r);
}
```



Example: swap pair

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
```

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  Store_field(r, 0, Field(p, 1));
  Store_field(r, 1, Field(p, 0));
  CAMLreturn(r);                    ⇐
}
```



Example: global variable

```c
value mem; // string

value caml_mem_init(value unit) {
  mem = caml_alloc_string(0);
  caml_register_global_root(&mem);
  return Val_unit;
}

value caml_mem_set(value str) {
  mem = str;
  return Val_unit;
}

value caml_mem_get(value unit) {
  return mem;
}
```

```
external mem_init : unit -> unit
  = "caml_mem_init"
external mem_get : unit -> string
  = "caml_mem_get"

length (mem_get ())
```

Example: global variable

```
value mem; // string

value caml_mem_init(value unit) {
  mem = caml_alloc_string(0);
  caml_register_global_root(&mem);
  return Val_unit;
}

value caml_mem_set(value str) {
  mem = str;
  return Val_unit;
}

value caml_mem_get(value unit) {
  return mem;
}
```



Potentially unsafe behaviour!



Example: global variable

```
value mem; // string

value caml_mem_init(value unit) {
  mem = caml_alloc_string(0);
  caml_register_global_root(&mem);
  return Val_unit;
}

value caml_mem_set(value str) {
  mem = str;
  return Val_unit;
}

value caml_mem_get(value unit) {
  return mem;
}
```

Exercises





Exercice: counter

```
value counter;  // int

value caml_counter_reset(value unit) {
  counter = 0;
  return Val_unit;
}

value caml_counter_get(value unit) {
  value result = counter;
  counter = counter + Val_int(1);
  return result;
}
```

Exercice: counter

```
value counter;  // int

value caml_counter_re
  counter = 0;
  return Val_unit;
}

value caml_counter_get(va
  value result = counte
  counter = counter +
  return result;
}
```



Exercice: counter

```
value counter; // int

value caml_counter_reset(value unit) {
  counter = Val_int(0);
  return Val_unit;
}

value caml_counter_get(value unit) {
  value result = counter;
  counter = Val_int(Int_val(counter) + 1);
  return result;
}
```

Exercice: swap variant

```
type ('a, 'b) either =
  | Left  of 'a
  | Right of 'b

external swap_variant : ('a, 'b) either -> ('b, 'a) either
  = "caml_swap_variant"
```

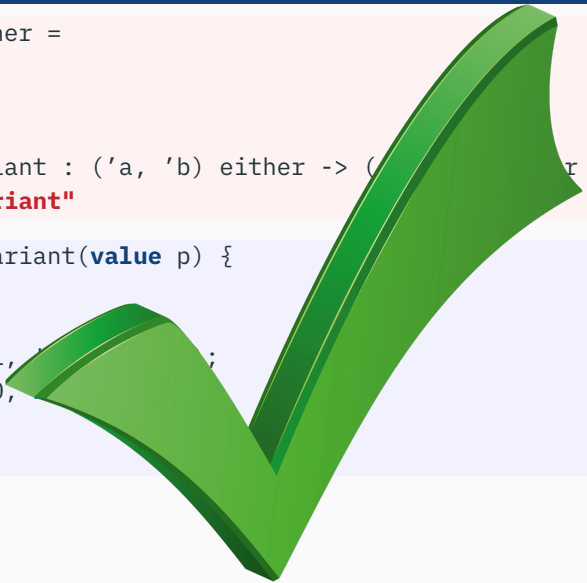
```
value caml_swap_variant(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(1, !Tag_val(p));
  Store_field(r, 0, Field(p, 0));
  CAMLreturn(r);
}
```

Exercice: swap variant

```
type ('a, 'b) either =
  | Left  of 'a
  | Right of 'b

external swap_variant : ('a, 'b) either -> (
  = "caml_swap_variant"
```

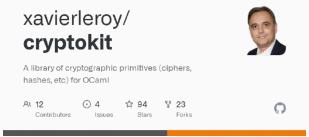
```
value caml_swap_variant(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(1,
  Store_field(r, 0,
  CAMLreturn(r);
}
```



Exercice: blake2b

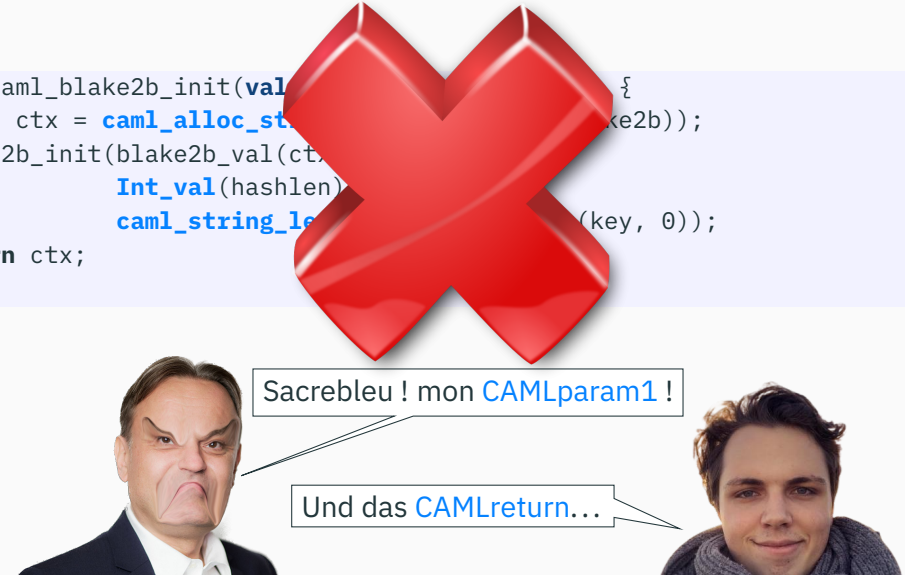
```
value caml_blake2b_init(value hashlen, value key) {
  value ctx = caml_alloc_string(sizeof(struct blake2b));
  blake2b_init(blake2b_val(ctx),
               Int_val(hashlen),
               caml_string_length(key), &Byte_u(key, 0));
  return ctx;
}
```





Exercice: blake2b

```
value caml_blake2b_init(val                                  {
  value ctx = caml_alloc_st                                  ke2b));
  blake2b_init(blake2b_val(ct
               Int_val(hashlen)
               caml_string_le                                (key, 0));
  return ctx;
}
```



Exercice: blake2b

```
value caml_blake2b_init(value hashlen, value key) {
  CAMLparam1(key);
  value ctx = caml_alloc_string(sizeof(struct blake2b));
  blake2b_init(blake2b_val(ctx),
               Int_val(hashlen),
               caml_string_length(key), &Byte_u(key, 0));
  CAMLreturn(ctx);
}
```



Melocoton



Melocoton: A Program Logic for Verified Interoperability Between OCaml and C

ARMAËL GUÉNEAU*, Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, Laboratoire Méthodes Formelles, France
JOHANNES HOSTERT*, Saarland University and MPI-SWS, Germany
SIMON SPIES*, MPI-SWS, Germany
MICHAEL SAMMLER, MPI-SWS, Germany
LARS BIRKEDAL, Aarhus University, Denmark
DEREK DREYER, MPI-SWS, Germany



Johannes Hostert
(JoJoDeveloping)



Armaël Guéneau



Simon Spies

Iris methodology



A Separation Logic Framework, implemented and verified in the Coq proof assistant

1. Define **language** (Λ) and its **operational semantics** ($\rightarrow$)
2. Define interpretation of **program state** in the Iris logic ($\_ \mapsto \_$)
3. Establish **proof rules** on expressions (`WP _ {Φ}`)

Iris methodology for Melocoton

Logic is built on top of **semantics**

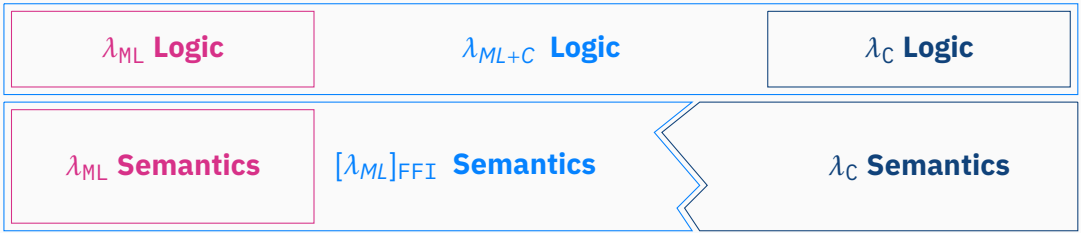


The main goal is to model the **communication** between OCaml and C $\lambda_{\mathrm{ML}} \oplus \lambda_{\mathrm{C}}$

But in practice they are too different

Iris methodology for Melocoton

Logic is built on top of **semantics**



The main goal is to model the **communication** between OCaml and C $\lambda_{\mathtt{ML}} \oplus \lambda_{\mathtt{C}}$

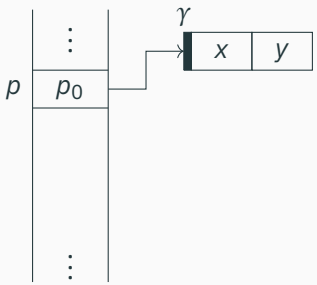
But in practice they are too different

Solution: Wrap $\lambda_{\mathtt{ML}}$ in $[-]_{\mathtt{FFI}}$ $[\lambda_{\mathtt{ML}}]_{\mathtt{FFI}} \oplus \lambda_{\mathtt{C}}$

Example: checking swap_pair

```
value caml_swap_pair(value p) {⇐
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
swap_pair (x, y)
```

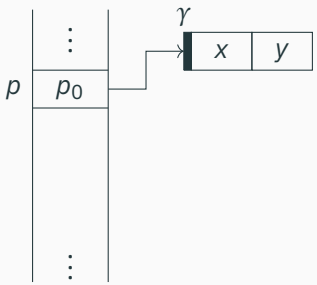


$\gamma \sim_{\texttt{ml}} (x,y)$

Example: checking swap_pair

```
value caml_swap_pair(value p) {⇐
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
swap_pair (x, y)
```



$p_0 \sim_\theta \gamma \sim_{\mathtt{ml}} (x,y)$

Resources

- $\mathrm{GC}\,\theta$
- $\gamma \mapsto_{\mathtt{blk}} [x;y]$
- $p \mapsto_{\mathtt{local}} p_0$

Properties

- $p_0 \sim_\theta \gamma$

Example: checking `swap_pair`

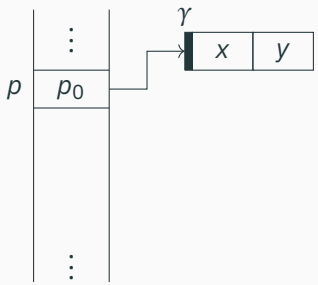
- $\mathrm{GC}\,\theta$: permission to use C functions of the FFI
  - θ: abstract name that identifies a **specific layout** of the GC memory
- $\gamma \mapsto_{\texttt{blk}} [x;y;\ldots]$: permission to access a block in the GC memory
  - γ: **abstract** label of the block
  - $[x;y;\ldots]$: content of the block
- $p \mapsto_{\texttt{local}} p_0$: permission to access the C variable p
  - p_0: the current value of the variable
- $p \mapsto_{\texttt{root}} \gamma$: Local rooted variable pointing to a GC label

---

C		FFI		ML
w	$\sim_\theta$	γ	$\sim$	v

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);                    ⇐
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```



$p_0 \sim_\theta \gamma \Rightarrow$ { $\mathrm{GC}\,\theta * p \mapsto_{\mathtt{local}} p_0$ }

CAMLparam1(p)

{ $\mathrm{GC}\,\theta * p \mapsto_{\mathtt{root}} \gamma$ }

Resources

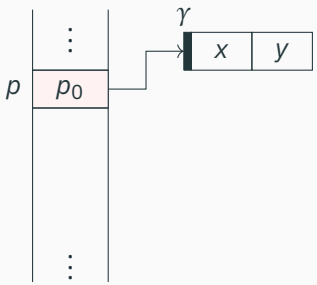
- $\mathrm{GC}\,\theta$
- $\gamma \mapsto_{\mathtt{blk}} [x;y]$
- $p \mapsto_{\mathtt{local}} p_0$

Properties

- $p_0 \sim_\theta \gamma$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);                    ⇐
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```



$p_0 \sim_\theta \gamma \Rightarrow$ { $\mathrm{GC}\,\theta * p \mapsto_{\mathtt{local}} p_0$ }

CAMLparam1(p)

{ $\mathrm{GC}\,\theta * p \mapsto_{\mathtt{root}} \gamma$ }

Resources

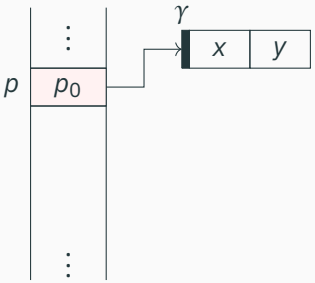
- $\mathrm{GC}\,\theta$
- $\gamma \mapsto_{\mathtt{blk}} [x;y]$
- $p \mapsto_{\mathtt{root}} \gamma$

Properties

- $p_0 \sim_\theta \gamma$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);                    ⇐
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```



$$\{ \quad \mathrm{GC}\,\theta \quad \}$$
$$\texttt{CAMLlocal1}(r)$$
$$\{ \quad \mathrm{GC}\,\theta * r \mapsto_{\mathtt{root}} () \quad \}$$

Resources

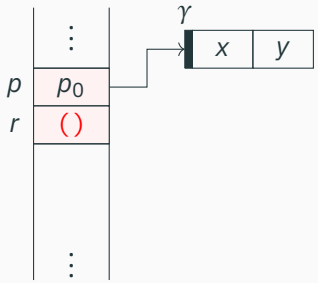
- $\mathrm{GC}\,\theta$
- $\gamma \mapsto_{\mathtt{blk}} [x;y]$
- $p \mapsto_{\mathtt{root}} \gamma$

Properties

- $p_0 \sim_\theta \gamma$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);                    ⇐
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```



{ GC θ }
CAMLlocal1(r)
{ GC θ ∗ r ↦root ( ) }

Resources

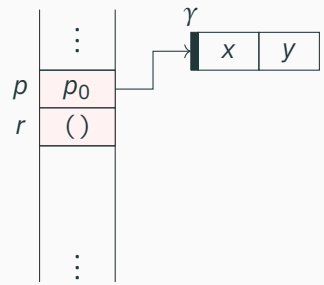
- GC θ
- γ ↦blk [x;y]
- p ↦root γ
- r ↦root ( )

Properties

- p0 ~θ γ

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```



$$\{\ \mathrm{GC}\,\theta\ \}$$
$$\texttt{caml\_alloc}(n,t)$$
$$\{\ \lambda r.\ \ \mathrm{GC}\,\theta' * \delta \mapsto_{\mathtt{blk}} [(); \ldots; ()] * r \sim_{\theta'} \delta\ \}$$

Resources

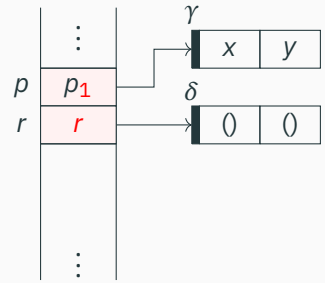
- $\mathrm{GC}\,\theta$
- $\gamma \mapsto_{\mathtt{blk}} [x; y]$
- $\delta \mapsto_{\mathtt{blk}} [;]$
- $p \mapsto_{\mathtt{root}} \gamma$
- $r \mapsto_{\mathtt{root}} ()$

Properties

- $p_0 \sim_\theta \gamma$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```



$$\{\ \mathrm{GC}\,\theta\ \}$$
$$\mathtt{caml\_alloc}(n,t)$$
$$\{\ \lambda r.\ \ \mathrm{GC}\,\theta' * \delta \mapsto_{\mathtt{blk}} [(); \ldots; ()] * r \sim_{\theta'} \delta\ \}$$

Resources

- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x; y]$
- $\delta \mapsto_{\mathtt{blk}} [(); ()]$
- $p \mapsto_{\mathtt{root}} \gamma$
- $r \mapsto_{\mathtt{root}} \delta$

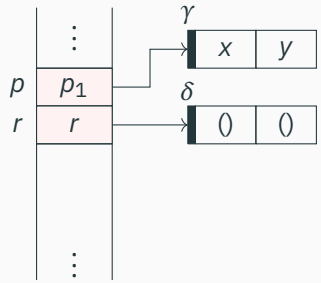
Properties

- $p_0 \sim_{\theta} \gamma$
- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

Example: checking `swap_pair`

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```

⇐



$$p \sim_\theta \gamma \Rightarrow \quad \{ \quad \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v_i; \ldots] \quad \} \; \mathtt{Field}(p, i) \; \{ \quad \lambda v_i.\ \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v_i; \ldots] \quad \}$$

Resources

- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x; y]$
- $\delta \mapsto_{\mathtt{blk}} [(); ()]$
- $p \mapsto_{\mathtt{root}} \gamma$
- $r \mapsto_{\mathtt{root}} \delta$

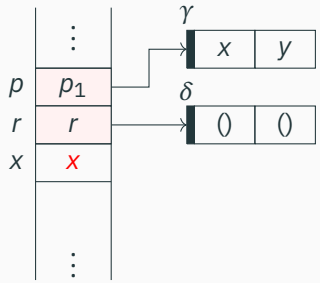
Properties

- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```

⇐



$$p \sim_\theta \gamma \Rightarrow \quad \{ \quad \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v_i; \ldots] \quad \} \; \mathtt{Field}(p,i) \; \{ \quad \lambda v_i.\ \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v_i; \ldots] \quad \}$$

Resources

- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x; y]$
- $\delta \mapsto_{\mathtt{blk}} [(); ()]$
- $p \mapsto_{\mathtt{root}} \gamma$
- $r \mapsto_{\mathtt{root}} \delta$

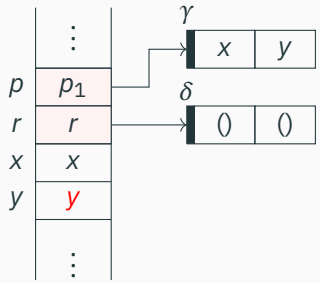
Properties

- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```

⇐



$$p \sim_\theta \gamma \Rightarrow \quad \{ \quad \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v_i; \ldots] \quad \} \;\; \mathtt{Field}(p, i) \;\; \{ \quad \lambda v_i.\ \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v_i; \ldots] \quad \}$$

Resources

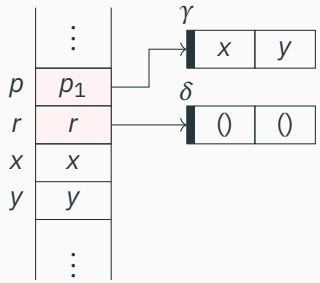
- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x; y]$
- $\delta \mapsto_{\mathtt{blk}} [(); ()]$
- $p \mapsto_{\mathtt{root}} \gamma$
- $r \mapsto_{\mathtt{root}} \delta$

Properties

- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

Example: checking `swap_pair`

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);          ⇐
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```



Resources

- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x; y]$
- $\delta \mapsto_{\mathtt{blk}} [(); ()]$
- $p \mapsto_{\mathtt{root}} \gamma$
- $r \mapsto_{\mathtt{root}} \delta$

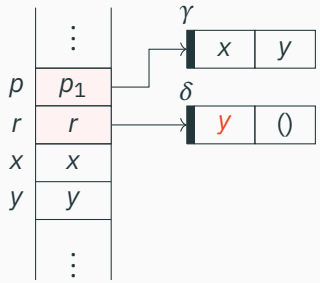
Properties

- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

$$p \sim_\theta \gamma \Rightarrow \quad \{ \quad \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v_i; \ldots] \quad \}$$
$$\mathtt{Store\_field}(p, i, v)$$
$$\{ \quad \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v; \ldots] \quad \}$$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);          ⇐
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```



Resources

- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x; y]$
- $\delta \mapsto_{\mathtt{blk}} [y; ()]$
- $p \mapsto_{\mathtt{root}} \gamma$
- $r \mapsto_{\mathtt{root}} \delta$

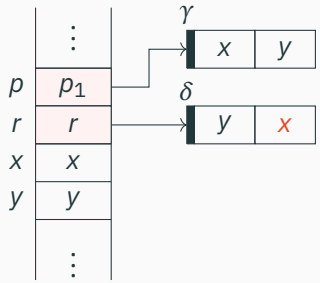
Properties

- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

$$p \sim_\theta \gamma \Rightarrow \quad \{\ \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v_i; \ldots]\ \}$$
$$\mathtt{Store\_field}(p, i, v)$$
$$\{\ \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v; \ldots]\ \}$$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);          ⇐
  CAMLreturn(r);
}
```



Resources

- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x;y]$
- $\delta \mapsto_{\mathtt{blk}} [y;x]$
- $p \mapsto_{\mathtt{root}} \gamma$
- $r \mapsto_{\mathtt{root}} \delta$

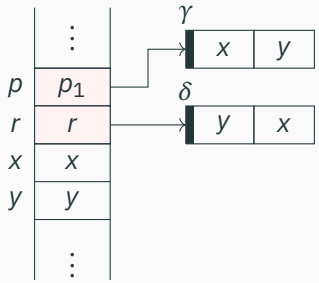
Properties

- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

$$p \sim_\theta \gamma \Rightarrow \quad \{\ \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v_i; \ldots]\ \}$$
$$\mathtt{Store\_field}(p, i, v)$$
$$\{\ \mathrm{GC}\,\theta * \gamma \mapsto_{\mathtt{blk}} [\ldots; v; \ldots]\ \}$$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);                    ⇐
}
```



$p_1 \sim_\theta \gamma \Rightarrow \ldots \Rightarrow$ { $\mathrm{GC}\,\theta * p \mapsto_{\mathtt{root}} \gamma * \ldots$ }

`CAMLreturn`(r)

{ $\mathrm{GC}\,\theta * p \mapsto_{\mathtt{local}} p_1 * \ldots$ }

Resources

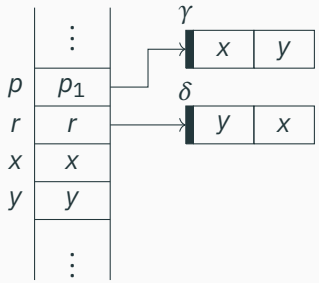
- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x;y]$
- $\delta \mapsto_{\mathtt{blk}} [y;x]$
- $p \mapsto_{\mathtt{root}} \gamma$
- $r \mapsto_{\mathtt{root}} \delta$

Properties

- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);                    ⇐
}
```



$p_1 \sim_\theta \gamma \Rightarrow \ldots \Rightarrow$ { $\mathrm{GC}\,\theta * p \mapsto_{\mathtt{root}} \gamma * \ldots$ }

`CAMLreturn`(r)

{ $\mathrm{GC}\,\theta * p \mapsto_{\mathtt{local}} p_1 * \ldots$ }

Resources

- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x;y]$
- $\delta \mapsto_{\mathtt{blk}} [y;x]$
- $p \mapsto_{\mathtt{local}} \gamma$
- $r \mapsto_{\mathtt{local}} \delta$

Properties

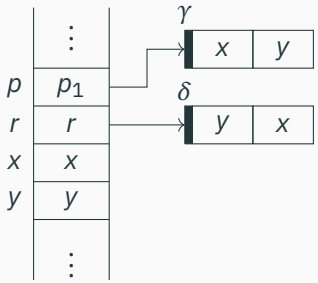
- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

Example: checking swap_pair

```
value caml_swap_pair(value p) {
  CAMLparam1(p);
  CAMLlocal1(r);
  r = caml_alloc(2, 0);
  value x = Field(p, 0);
  value y = Field(p, 1);
  Store_field(r, 0, y);
  Store_field(r, 1, x);
  CAMLreturn(r);
}
```

```
external swap_pair :
  'a * 'b -> 'b * 'a =
  "caml_swap_pair"
swap_pair (x, y)
```

⇐



$p_1 \sim_{\theta'} \gamma \sim_{\mathtt{ml}} (x,y)$

$r \sim_{\theta'} \delta \sim_{\mathtt{ml}} (y,x)$

Resources

- $\mathrm{GC}\,\theta'$
- $\gamma \mapsto_{\mathtt{blk}} [x;y]$
- $\delta \mapsto_{\mathtt{blk}} [y;x]$
- $p \mapsto \gamma$
- $r \mapsto \delta$

Properties

- $p_1 \sim_{\theta'} \gamma$
- $r \sim_{\theta'} \delta$

Showtime





Conclusion



https://melocoton-project.github.io/