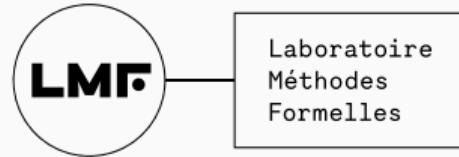
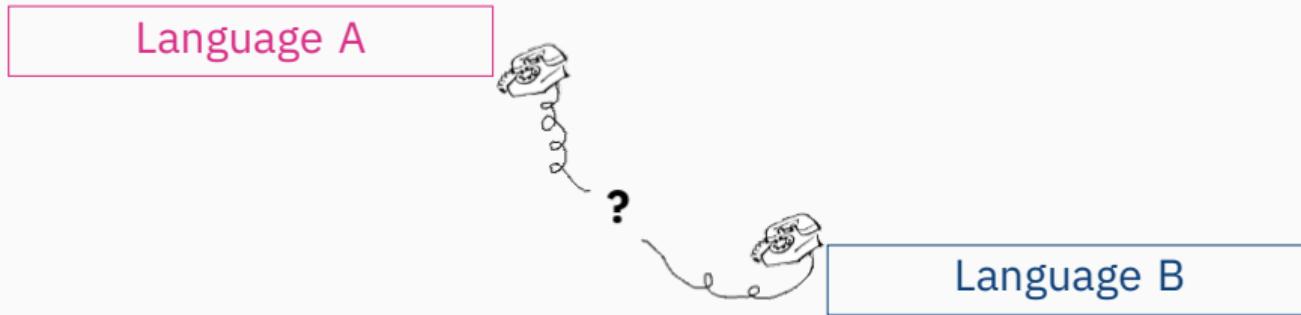


Formalisation of interoperability between C and OCaml

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



```
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

dbuenzli/tsdl

Thin bindings to SDL for OCaml

Languages



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;  
}
```

-	-	-	-	-	-	-	-	1
---	---	---	---	---	---	---	---	---

Integer

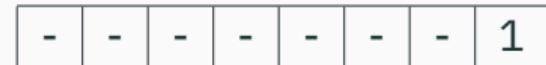
-	-	-	-	-	-	-	-	0
---	---	---	---	---	---	---	---	---

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

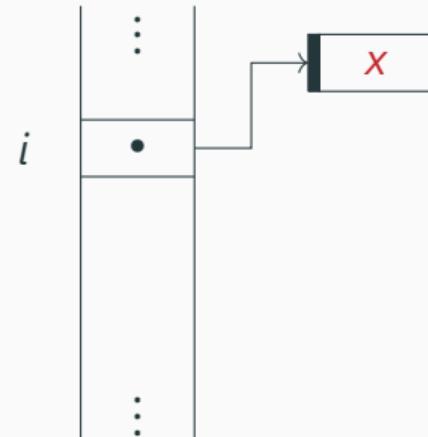
```
external incr : int ref -> unit =
  "caml_incr"
```

```
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"

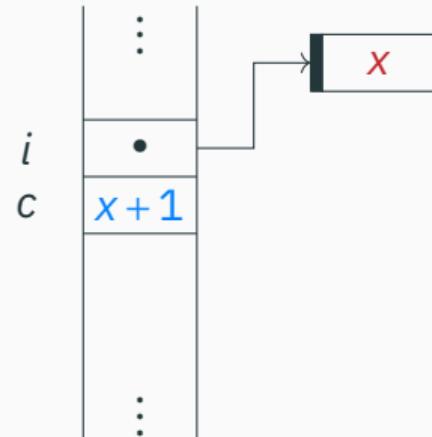
value caml_incr(value i) {           <=
  int c = 1 + Int_val(Field(i, 0));
  Store_field(i, 0, Val_int(c));
  return Val_unit;
}
```



Example: increment

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

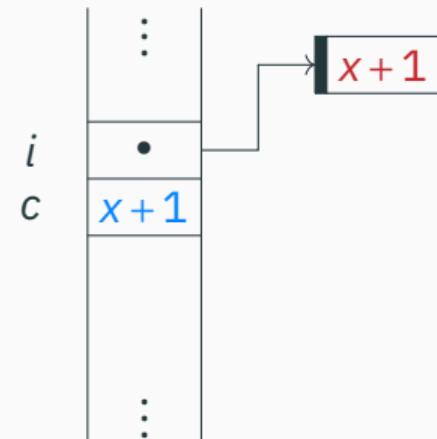
```
value caml_incr(value i) {
  int c = 1 + Int_val(Field(i, 0));      ←
  Store_field(i, 0, Val_int(c));
  return Val_unit;
}
```



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;
}
```

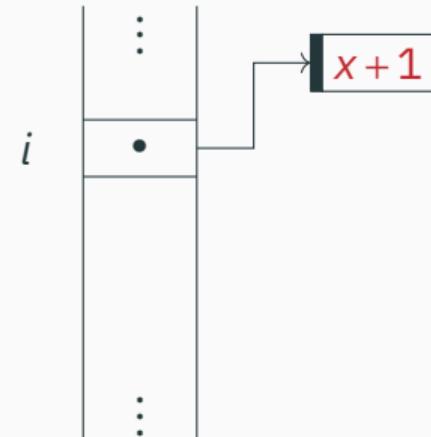


Example: increment

```
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

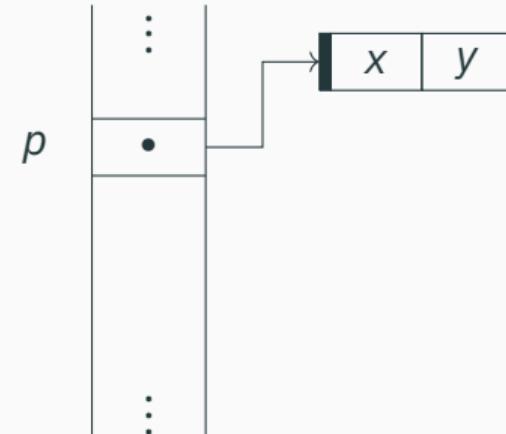
```
external swap_pair :  
  'a * 'b -> 'b * 'a =  
  "caml_swap_pair"
```

```
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"
```

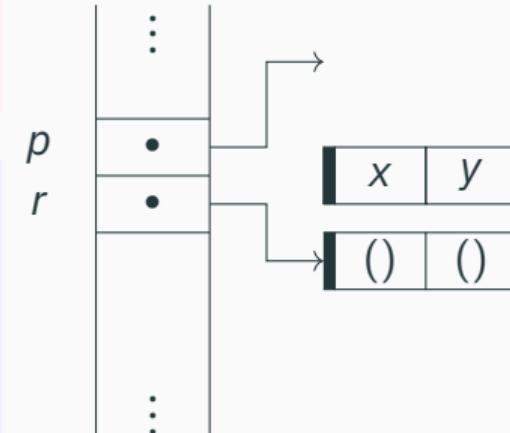
```
value caml_swap_pair(value p) {  ←  
  value r = caml_alloc(2, 0);  
  Store_field(r, 0, Field(p, 1));  
  Store_field(r, 1, Field(p, 0));  
  return r;  
}
```



Example: swap pair

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

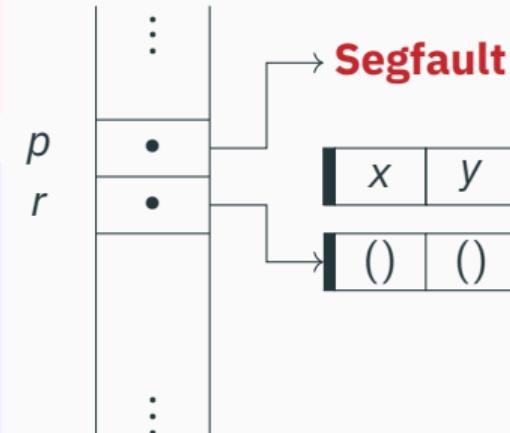
```
value caml_swap_pair(value p) {  
  value r = caml_alloc(2, 0);    ←  
  Store_field(r, 0, Field(p, 1));  
  Store_field(r, 1, Field(p, 0));  
  return r;  
}
```



Example: swap pair

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

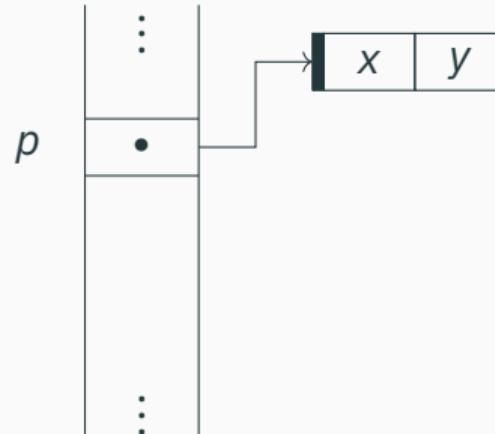
```
value caml_swap_pair(value p) {  
  value r = caml_alloc(2, 0);  
  Store_field(r, 0, Field(p, 1)); <=  
  Store_field(r, 1, Field(p, 0));  
  return r;  
}
```



Example: swap pair

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

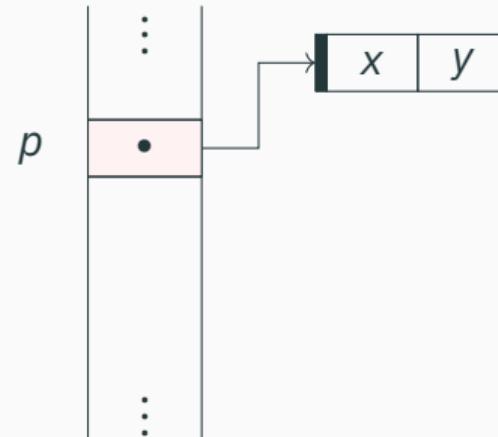
```
value caml_swap_pair(value p) {  ←  
  CAMLparam1(p);  
  CAMLlocal1(r);  
  r = caml_alloc(2, 0);  
  Store_field(r, 0, Field(p, 1));  
  Store_field(r, 1, Field(p, 0));  
  CAMLreturn(r);  
}
```



Example: swap pair

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

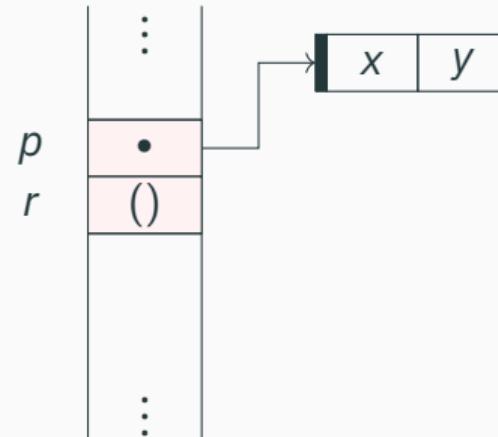
```
value caml_swap_pair(value p) {  
  CAMLparam1(p);           ←  
  CAMLlocal1(r);  
  r = caml_alloc(2, 0);  
  Store_field(r, 0, Field(p, 1));  
  Store_field(r, 1, Field(p, 0));  
  CAMLreturn(r);  
}
```



Example: swap pair

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

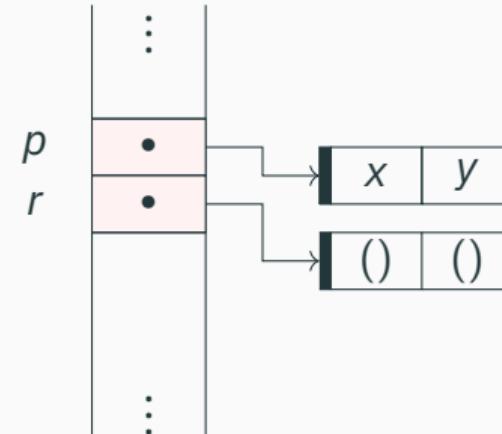
```
value caml_swap_pair(value p) {  
  CAMLparam1(p);  
  CAMLlocal1(r);  
  ←  
  r = caml_alloc(2, 0);  
  Store_field(r, 0, Field(p, 1));  
  Store_field(r, 1, Field(p, 0));  
  CAMLreturn(r);  
}
```



Example: swap pair

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

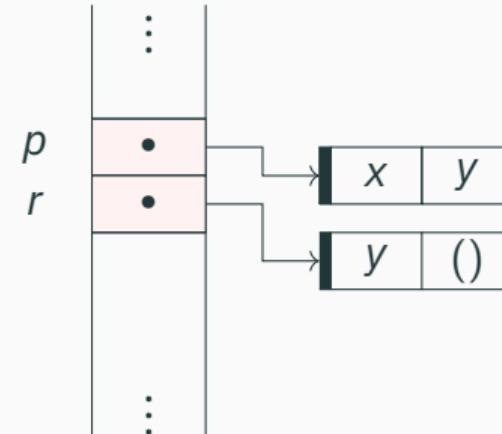
```
value caml_swap_pair(value p) {  
  CAMLparam1(p);  
  CAMLlocal1(r);  
  r = caml_alloc(2, 0);           ←  
  Store_field(r, 0, Field(p, 1));  
  Store_field(r, 1, Field(p, 0));  
  CAMLreturn(r);  
}
```



Example: swap pair

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

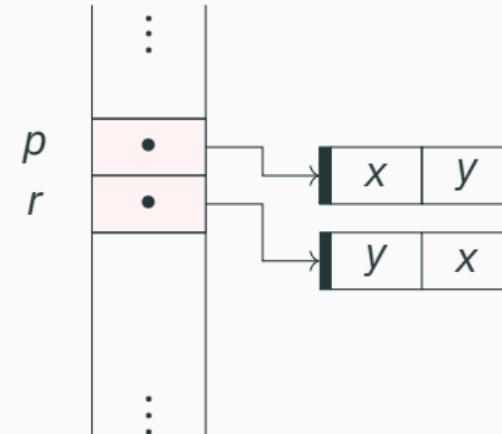
```
value caml_swap_pair(value p) {  
  CAMLparam1(p);  
  CAMLlocal1(r);  
  r = caml_alloc(2, 0);  
  Store_field(r, 0, Field(p, 1)); <=  
  Store_field(r, 1, Field(p, 0));  
  CAMLreturn(r);  
}
```



Example: swap pair

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

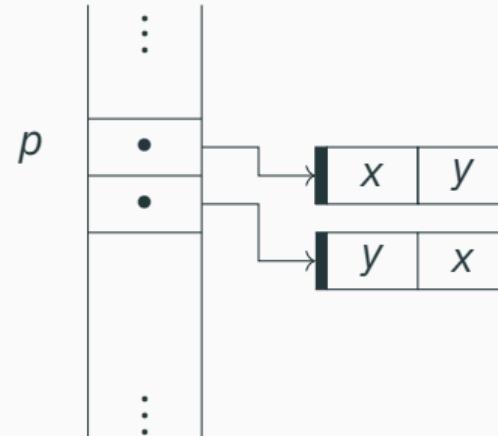
```
value caml_swap_pair(value p) {  
  CAMLparam1(p);  
  CAMLlocal1(r);  
  r = caml_alloc(2, 0);  
  Store_field(r, 0, Field(p, 1));  
  Store_field(r, 1, Field(p, 0)); <=  
  CAMLreturn(r);  
}
```



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

```
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;
}

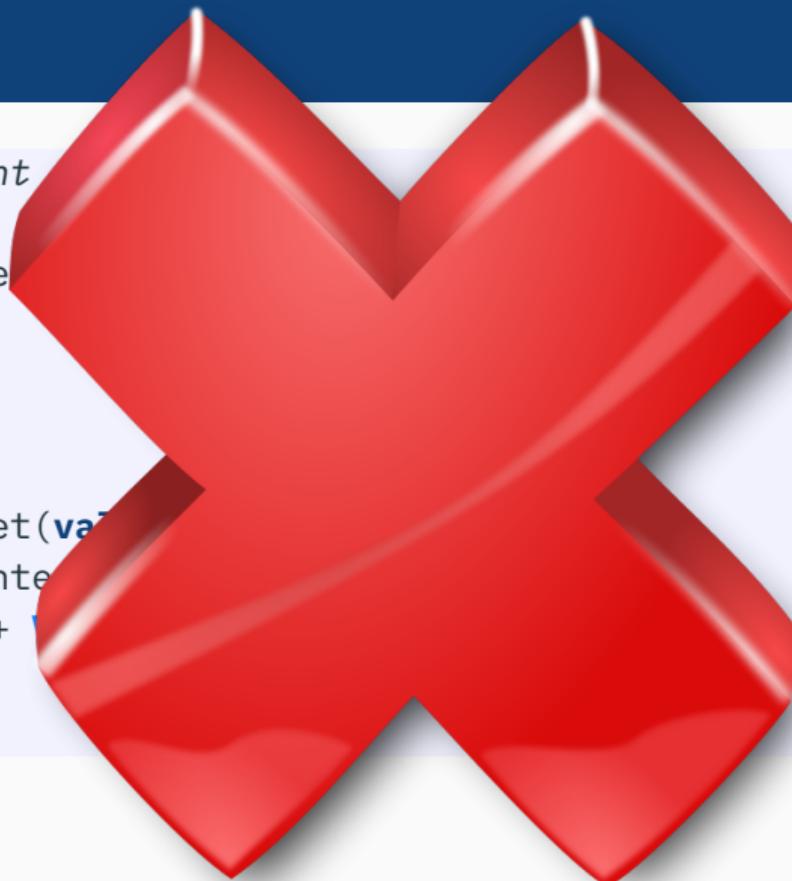
value caml_counter_get(value unit) {
    value result = counter;
    counter = counter + Val_int(1);
    return result;
}
```

Exercice: counter

```
value counter; // int

value caml_counter_re
    counter = 0;
    return Val_unit;
}

value caml_counter_get(va
    value result = counter;
    counter = counter + 1;
    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 -> ('b, 'a) either
= "caml_swap_variant"
```

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



Téma le code !

xavierleroy/
cryptokit

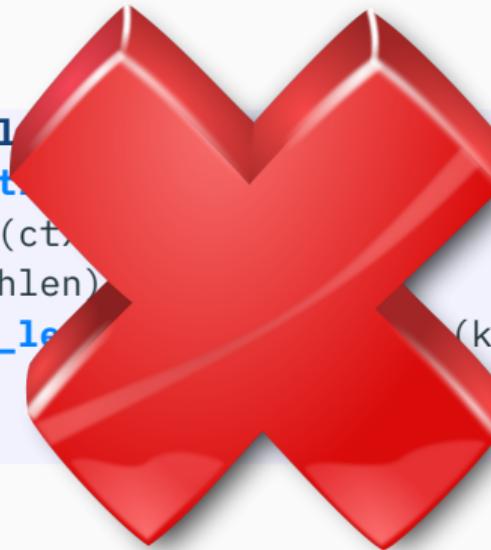
A library of cryptographic primitives (ciphers,
hashes, etc) for OCaml

12 Contributors 4 Issues 94 Stars 23 Forks



Exercice: blake2b

```
value caml_blake2b_init(value param1)
  value ctx = caml_alloc_st...
    blake2b_init(blake2b_val(ct...
      Int_val(hashlen))
      caml_string_le...
        (key, 0));
  return ctx;
}
```



Sacrebleu ! mon CAMLparam1 !

Und das CAMLreturn...



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);  
}
```



Merci Simon !



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)

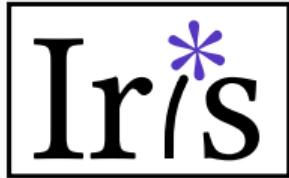


Armaël Guéneau



Simon Spies

Iris methodology

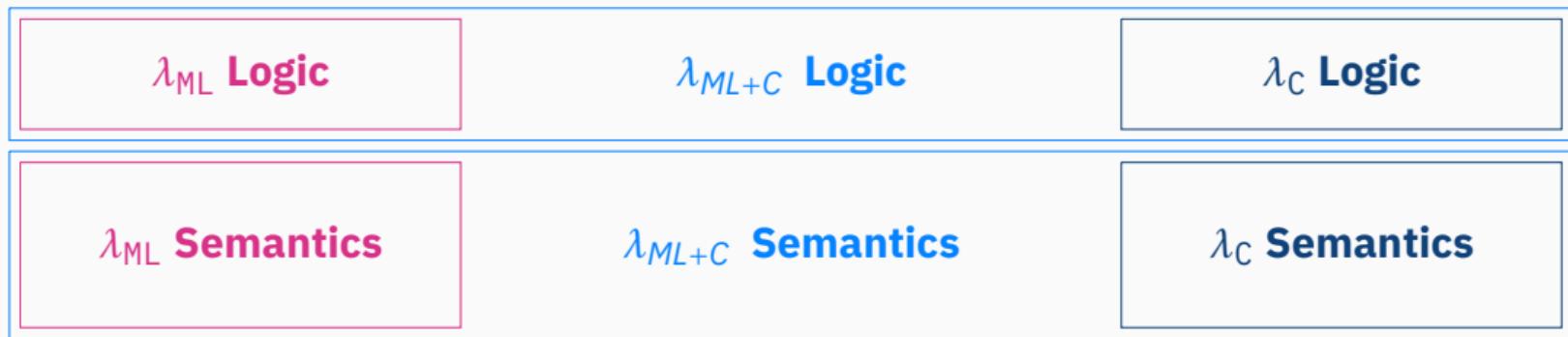


A Separation Logic Framework, implemented
and verified in the Coq proof assistant

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

Iris methodology for Melocoton

Logic is built on top of **semantics**



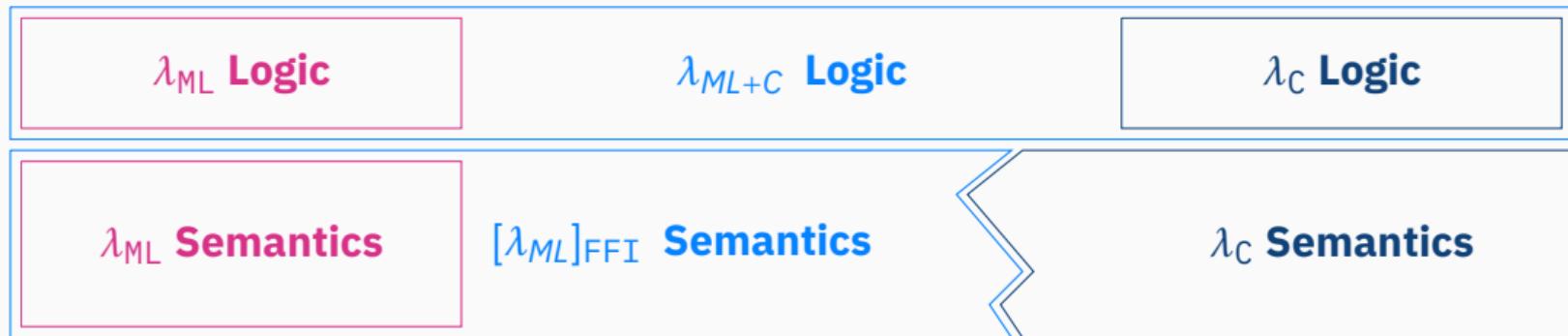
The main goal is to model the **communication** between OCaml and C

$$\lambda_{ML} \oplus \lambda_C$$

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

$$\lambda_{ML} \oplus \lambda_C$$

But in practice they are too different

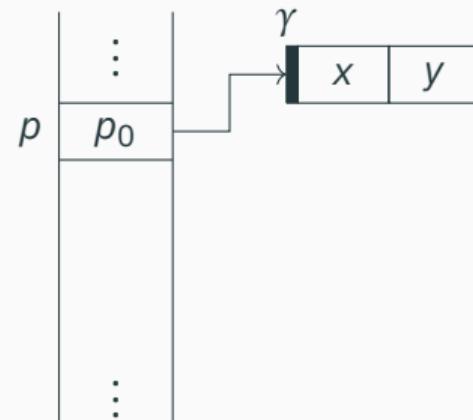
Solution: Wrap λ_{ML} in $[-]_{FFI}$

$$[\lambda_{ML}]_{FFI} \oplus \lambda_C$$

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)
```

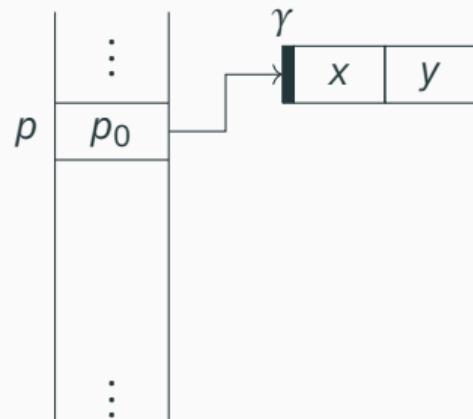


$$\gamma \sim_{\text{ml}} (x, y)$$

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)
```



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

Resources

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

Properties

- $p_0 \sim_{\theta} \gamma$

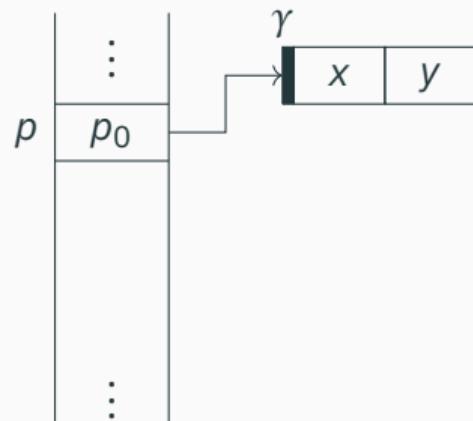
Example: checking swap_pair

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

$$\begin{array}{ccc} \text{C} & \text{FFI} & \text{ML} \\ w & \sim_\theta & \gamma \\ & & \sim & v \end{array}$$

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);  
}
```



$$p_0 \sim_{\theta} \gamma \Rightarrow \{ \begin{array}{l} \text{GC } \theta * p \mapsto_{\text{local}} p_0 \\ \text{CAMLparam1}(p) \end{array} \} \quad \{ \begin{array}{l} \text{GC } \theta * p \mapsto_{\text{root}} \gamma \end{array} \}$$

Resources

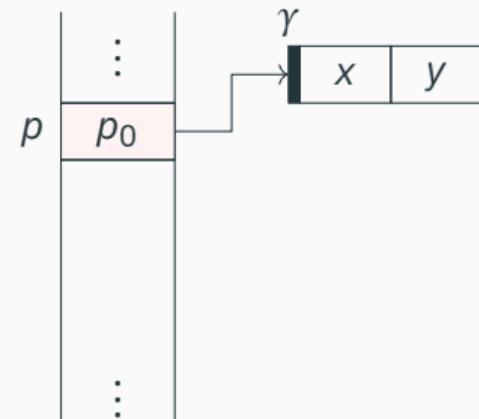
- $\text{GC } \theta$
- $\gamma \mapsto_{\text{blk}} [x; y]$
- $p \mapsto_{\text{local}} p_0$

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);  
}
```



$$p_0 \sim_{\theta} \gamma \Rightarrow \{ \begin{array}{l} \text{GC } \theta * p \mapsto_{\text{local}} p_0 \\ \text{CAMLparam1}(p) \end{array} \} \quad \{ \text{GC } \theta * p \mapsto_{\text{root}} \gamma \}$$

Resources

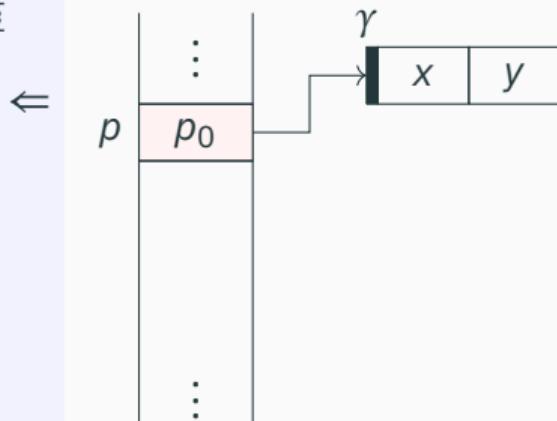
- $GC \theta$
- $\gamma \mapsto_{blk} [x; y]$
- $p \mapsto_{root} \gamma$

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);  
}
```



Resources

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

Properties

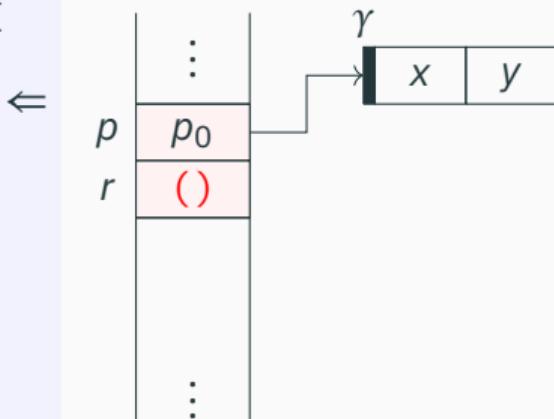
- $p_0 \sim_\theta \gamma$

```
{           GCθ          }  
    CAMLlocal1(r)  
{  GCθ * r → root () }
```

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);  
}
```

```
{           GCθ          }  
    CAMLlocal1(r)  
{   GCθ * r→root () }
```



Resources

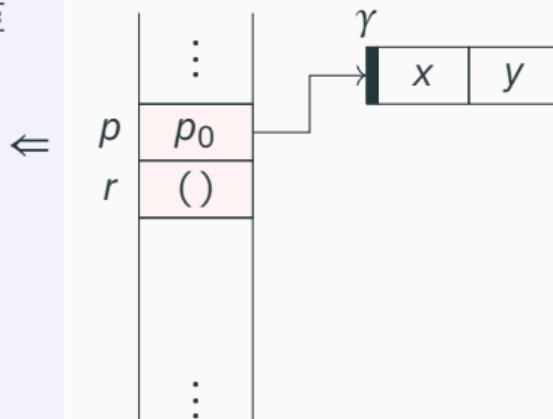
- $GC\theta$
- $\gamma \mapsto_{blk} [x; y]$
- $p \mapsto_{root} \gamma$
- $r \mapsto_{root} ()$

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);  
}
```



```
{ GCθ  
      caml_alloc(n,t)  
{ λr. GCθ' * δ →blk [(());...;()] * r ~θ' δ }
```

Resources

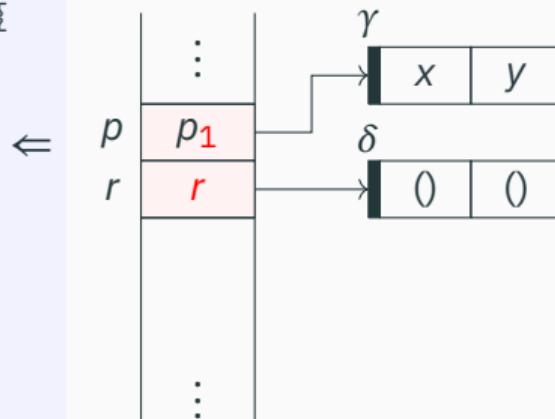
- $GC\theta$
- $\gamma \mapsto_{blk} [x;y]$
- $\delta \mapsto_{blk} []$
- $p \mapsto_{root} \gamma$
- $r \mapsto_{root} ()$

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);  
}
```



```
{ GCθ  
      caml_alloc(n,t)  
{ λr. GCθ' * δ →blk [( ); ...; ()] * r ~θ' δ }
```

Resources

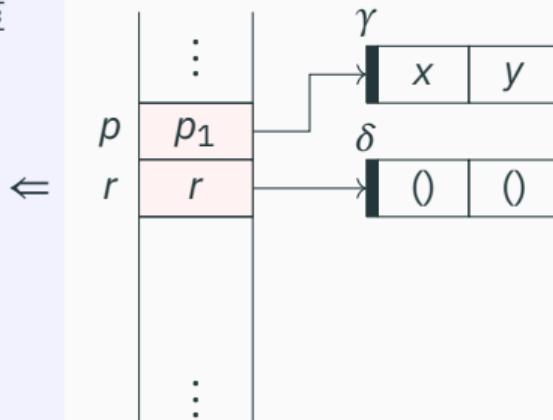
- $GC\theta'$
- $\gamma \rightarrow_{blk} [x; y]$
- $\delta \rightarrow_{blk} [(); ()]$
- $p \rightarrow_{root} \gamma$
- $r \rightarrow_{root} \delta$

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);  
}
```



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

Resources

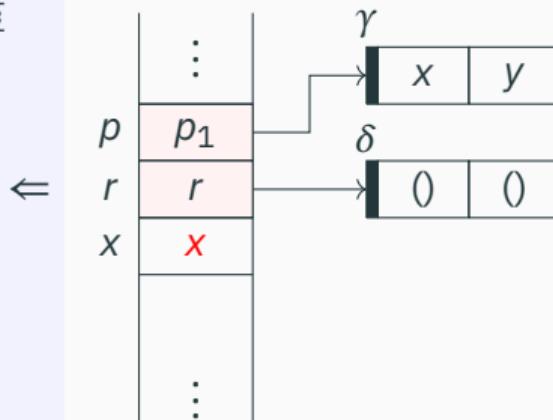
- $\text{GC } \theta'$
- $\gamma \mapsto_{\text{blk}} [x; y]$
- $\delta \mapsto_{\text{blk}} [(); ()]$
- $p \mapsto_{\text{root}} \gamma$
- $r \mapsto_{\text{root}} \delta$

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);  
}
```



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

Resources

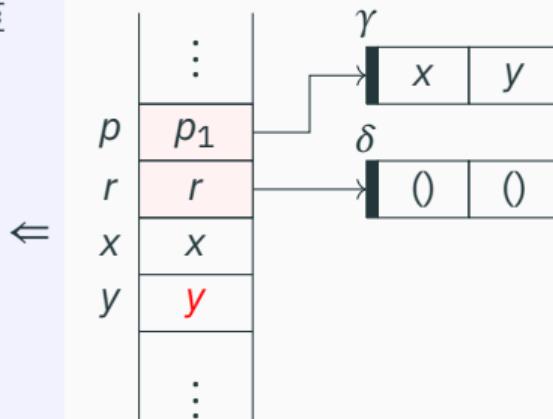
- $p_1 \sim_{\theta'} \gamma$
- $\gamma \mapsto_{\text{blk}} [x; y]$
- $\delta \mapsto_{\text{blk}} [(0); (0)]$
- $p \mapsto_{\text{root}} \gamma$
- $r \mapsto_{\text{root}} \delta$

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);
}
```



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

Resources

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

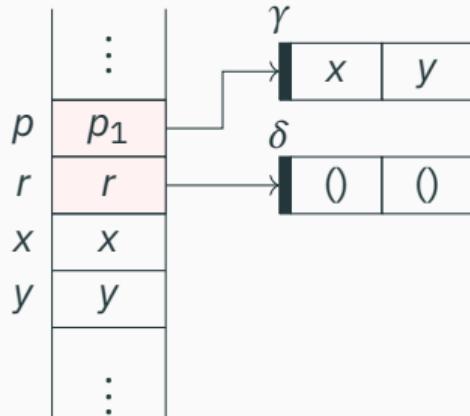
Properties

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

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);  
}
```

\Leftarrow



$$p \sim_{\theta} \gamma \Rightarrow \{ \text{GC } \theta * \gamma \mapsto_{\text{blk}} [\dots; v_i; \dots] \} \\ \quad \text{Store_field}(p, i, v) \\ \{ \text{GC } \theta * \gamma \mapsto_{\text{blk}} [\dots; v; \dots] \}$$

Resources

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

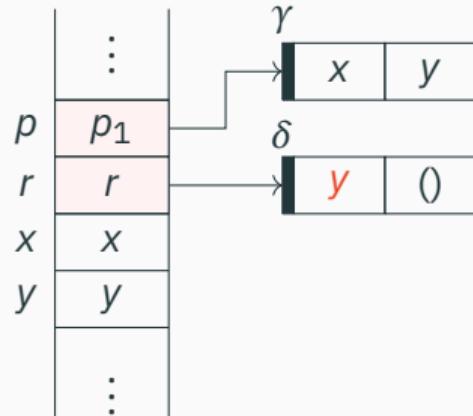
Properties

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

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);  
}
```

\Leftarrow



$$p \sim_{\theta} \gamma \Rightarrow \{ \text{GC } \theta * \gamma \rightarrow_{\text{blk}} [\dots; v_i; \dots] \} \\ \quad \text{Store_field}(p, i, v) \\ \{ \text{GC } \theta * \gamma \rightarrow_{\text{blk}} [\dots; v; \dots] \}$$

Resources

- $\text{GC } \theta'$
- $\gamma \rightarrow_{\text{blk}} [x; y]$
- $\delta \rightarrow_{\text{blk}} [y; ()]$
- $p \rightarrow_{\text{root}} \gamma$
- $r \rightarrow_{\text{root}} \delta$

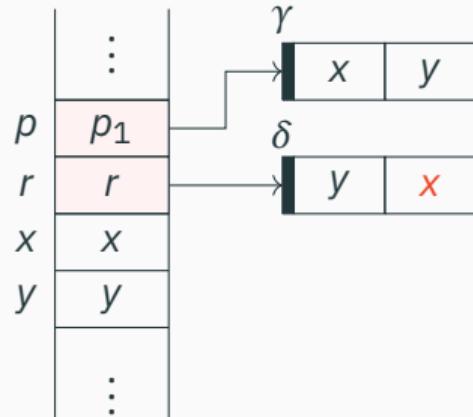
Properties

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

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);  
}
```

⇐



$$p \sim_{\theta} \gamma \Rightarrow \{ \text{GC } \theta * \gamma \rightarrow_{\text{blk}} [\dots; v_i; \dots] \} \\ \quad \text{Store_field}(p, i, v) \\ \{ \text{GC } \theta * \gamma \rightarrow_{\text{blk}} [\dots; v; \dots] \}$$

Resources

- $\text{GC } \theta'$
- $\gamma \rightarrow_{\text{blk}} [x; y]$
- $\delta \rightarrow_{\text{blk}} [y; x]$
- $p \rightarrow_{\text{root}} \gamma$
- $r \rightarrow_{\text{root}} \delta$

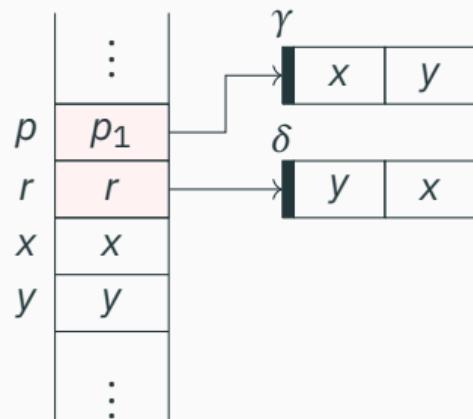
Properties

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

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);  
}
```

⇐



$$p_1 \sim_{\theta} \gamma \Rightarrow \dots \Rightarrow \{ \quad \text{GC } \theta * p \xrightarrow{\text{root}} \gamma * \dots \quad \}$$

$\text{CAMLreturn}(r)$

$$\{ \quad \text{GC } \theta * p \xrightarrow{\text{local}} p_1 * \dots \quad \}$$

Resources

- $\text{GC } \theta'$
- $\gamma \xrightarrow{\text{blk}} [x; y]$
- $\delta \xrightarrow{\text{blk}} [y; x]$
- $p \xrightarrow{\text{root}} \gamma$
- $r \xrightarrow{\text{root}} \delta$

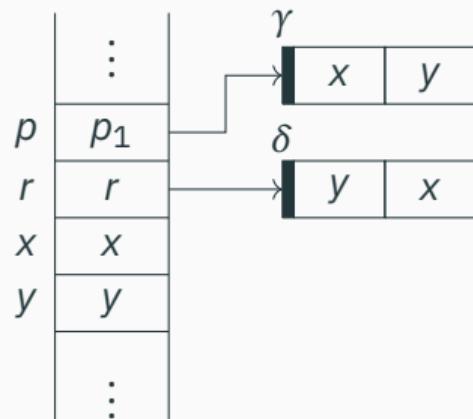
Properties

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

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);  
}
```

⇐



$$p_1 \sim_{\theta} \gamma \Rightarrow \dots \Rightarrow \{ \quad \text{GC } \theta * p \rightarrow_{\text{root}} \gamma * \dots \quad \}$$

CAMLreturn(r)

$$\{ \quad \text{GC } \theta * p \rightarrow_{\text{local}} p_1 * \dots \quad \}$$

Resources

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

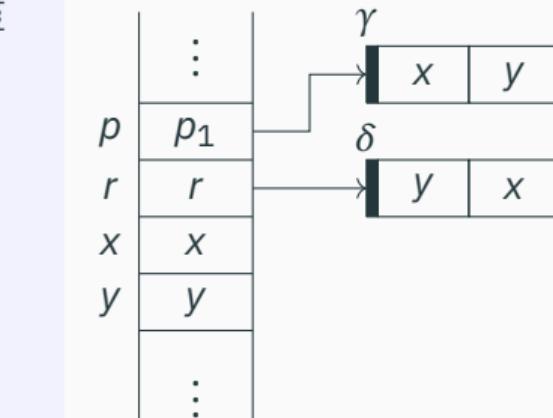
Properties

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

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)
```



$$\begin{aligned} p_1 &\sim_{\theta'} \gamma \sim_{\text{ml}} (x, y) \\ r &\sim_{\theta'} \delta \sim_{\text{ml}} (y, x) \end{aligned}$$

\Leftarrow

Resources

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

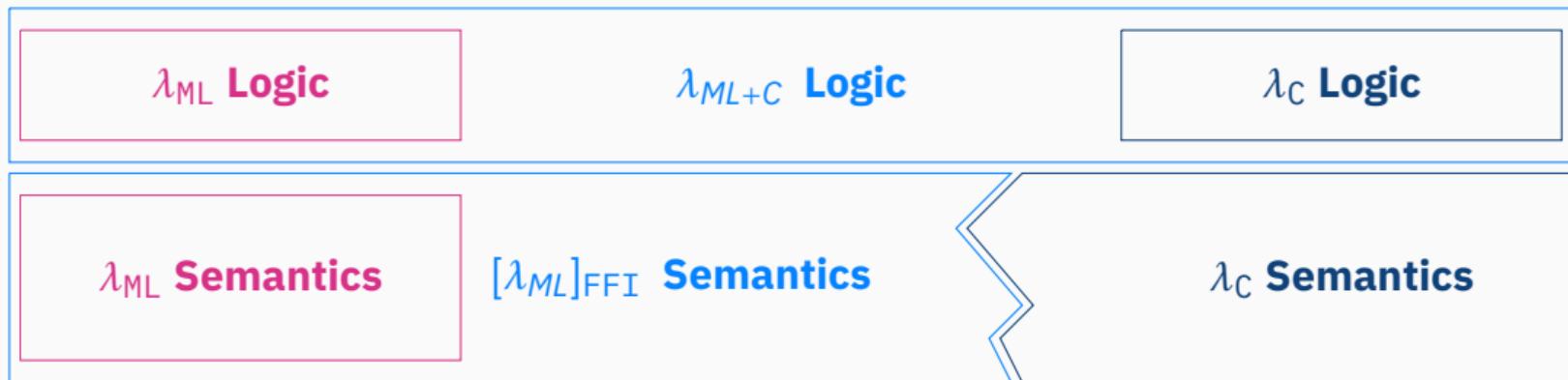
Properties

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

Showtime



Conclusion



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