Issue VII: Symmetric Interpreter

Maksym Sokhatskyi [1]

[1] National Technical University of Ukraine
Igor Sikorsky Kyiv Polytechnical Institute
15 травня 2025 р.

Анотація

Minimal language for parallel computations in symmetric monoidal categories.
Keywords: Interaction Networks, Symmetric Monoidal Categories

Зміст

Присвячується автору
Interaction Networks
Combinators

Іву Лафону

1 The Yves Language

Мова програмування **Yves** — це внутрішня мова симетричних моноїдальних категорій, що реалізує паралельні обчислення через взаємодію комбінаторів (ζ, δ, ϵ) з правилами анігіляції та комутації, придатна для моделювання лінійних і паралельних систем.

1.1 Синтаксис

Definition 1. Терми **Yves** складаються зі змінних, комбінаторів (Con, Dup, Era), пар, обміну (Swap), зв'язування (Let) та одиниці (Unit). Мова підтримує афінну логіку, забороняючи повторне використання змінних.

```
I = #identifier
Y = I | Con Y  | Dup Y | Era Y
      | Pair (Y, Y)    | Unit
      | Let (I, Y, Y)  | Swap Y
```

Definition 2. Кодування термів у мові OCaml:

```
type term =
    | Var of string
    | Con of term
    | Dup of term
    | Era of term
    | Pair of term * term
    | Swap of term
    | Let of string * term * term
    | Unit
```

1.2 Семантика

Theorem 1. Правила обчислень у **Yves** базуються на анігіляції та комутації комбінаторів:

```
Con (Con x) → Pair (x, x)
Dup (Dup x) → Pair (x, x)
Era (Era x) → Unit
Con (Dup x) → Dup (Con x)
Con (Era x) → Pair (Era x, Era x)
Dup (Era x) → Pair (Era x, Era x)
Swap (Pair (t, u)) → Pair (u, t)
Let (x, t, u) → subst x t u
```

$$\frac{\zeta(\zeta(x))}{(x,x)} \qquad (\zeta\text{-annihilation})$$

$$\frac{\delta(\delta(x))}{(x,x)} \qquad (\delta\text{-annihilation})$$

$$\frac{\varepsilon(\varepsilon(x))}{\mathbf{1}} \qquad (\epsilon\text{-annihilation})$$

1.2.1 Підстановка

Definition 3. Підстановка в **Yves**:

```
let rec subst env var term = function
    | Var v ->
        if v = var then
            if is_bound var env then failwith "Affine violation: variable used twice"
            else term
        else Var v
    | Con t -> Con (subst env var term t)
    | Dup t -> Dup (subst env var term t)
    | Era t -> Era (subst env var term t)
    | Pair (t, u) -> Pair (subst env var term t, subst env var term u)
    | Swap t -> Swap (subst env var term t)
    | Let (x, t1, t2) ->
        let t1' = subst env var term t1 in
        if x = var then Let (x, t1', t2)
        else Let (x, t1', subst env var term t2)
    | Unit -> Unit
```

1.2.2 Редукція

Definition 4. Редукція термів у **Yves**:

```
let reduce env term =
    match term with
    | Con (Con x) -> Pair (x, x)
    | Dup (Dup x) -> Pair (x, x)
    | Era (Era x) -> Unit
    | Con (Dup x) -> Dup (Con x)
    | Con (Era x) -> Pair (Era x, Era x)
    | Dup (Era x) -> Pair (Era x, Era x)
    | Swap (Pair (t, u)) -> Pair (u, t)
    | Let (x, t, u) -> subst env x t u
    | _ -> term
```

1.2.3 Пошук пар

Definition 5. Пошук активних пар для редукції:

```
let rec find_redexes env term acc =
    match term with
    | Con (Con x) -> (term, Pair (x, x)) :: acc
    | Dup (Dup x) -> (term, Pair (x, x)) :: acc
    | Era (Era x) -> (term, Unit) :: acc
    | Con (Dup x) -> (term, Dup (Con x)) :: acc
    | Con (Era x) -> (term, Pair (Era x, Era x)) :: acc
    | Dup (Era x) -> (term, Pair (Era x, Era x)) :: acc
    | Swap (Pair (t, u)) -> (term, Pair (u, t)) :: acc
    | Let (x, t, u) -> (term, subst env x t u) :: find_redexes env t (find_redexes env u acc)
    | Con t ->
      (match t with
       | Dup _ | Era _ -> acc
       | Con x -> find_redexes env t ((term, reduce env term) :: acc)
       | _ -> find_redexes env t acc)
    | Dup t -> find_redexes env t acc
    | Era t -> find_redexes env t acc
    | Pair (t, u) ->
      let acc' = find_redexes env t acc in
      find_redexes env u acc'
    | Swap t -> find_redexes env t acc
    | Var _ | Unit -> acc
```

1.2.4 Паралельна редукція

Definition 6. Паралельна редукція:

```
let eval_parallel pool env term =
    let rec loop term =
        let redexes = find_redexes env term [] in
        if redexes = [] then term
        else
            let new_term = Task.run pool (fun () ->
                List.fold_left
                    (fun acc (old_t, new_t) -> replace_subterm old_t new_t acc)
                    term redexes
            ) in
            loop new_term
    in
    loop term
```

1.2.5 Внутрішня мова CMK

Theorem 2. Доведення, що мова **Yves** є внутрішньою мовою симетричних моноїдальних категорій:

$$\begin{cases} \text{Let} : A \to C(u \cdot t, t : A \to B, u : B \to C), \\ \text{Pair} : A \to B \to A \otimes B, \\ \text{Swap} : A \otimes B \to B \otimes A, \\ \text{Con} : A \otimes A \to A, \\ \text{Dup} : A \to A \otimes A, \\ \text{Era} : A \to \mathbf{1}, \\ \text{Var} : A, \\ \text{Unit} : \mathbf{1}. \end{cases}$$

Конструктори відповідають аксіомам CMK:

- Pair моделює тензорний добуток $\otimes$. - Swap реалізує симетрію $\sigma_{A,B}$ з умовою $\sigma_{B,A} \circ \sigma_{A,B} = \text{id}_{A \otimes B}$. - Unit є одиничним об'єктом I для якого $A \otimes I \cong A$. - Let моделює композицію морфізмів (асоціативність). - Dup та Era утворюють структуру комоноїда. - Con діє як контракція.

Лямбда-функція і аплікація:

$$\begin{cases} \lambda x.t \vdash \text{Con}(\text{Let}(x, \text{Var}(x), t)), \\ tu \mapsto \text{Con}(\text{Pair}(t, u)). \end{cases}$$