ADT4HPC: Algebraic Data Types for High Performance Computing

Journées Nationales du GDR GPL, Juin 2022

Thaïs Baudon (https://perso.ens-lyon.fr/thais.baudon/) – ENS de Lyon, France

Ph.D. advisors: Laure Gonnord, Gabriel Radanne

Algebraic Data Types: what for?

Type declaration example: red-black trees (RBT)

Color = *Red* + *Black*

RBT = *Empty* + *Node* (Color, i32, &RBT, &RBT)

- Type-safety, non-redundancy and exhaustivity checks.
- Concise expression of complex functions through **pattern-matching**.

Pattern matching on RBTs: example

```
match c, v, t1, t2 {
  Black,z, &Node(Red,y, &Node(Red,x,a,b),c), d
  | Black,z, &Node(Red,x,a, &Node(Red,y,b,c)), d
  | Black,x,a, &Node(Red,z, &Node(Red,y,b,c),d)
  | Black,x,a, &Node(Red,y,b, &Node(Red,z,c,d))
  => Node(Red, y, &Node(Black,x,a,b), &Node(Black,z,c,d)),
 a, b, c, d => Node (a, b, c, d),
}
```

- Available in different kinds of languages: OCaml, Haskell, Scala, Rust…
- Yet complex structures are used in HPC *without ADTs*!

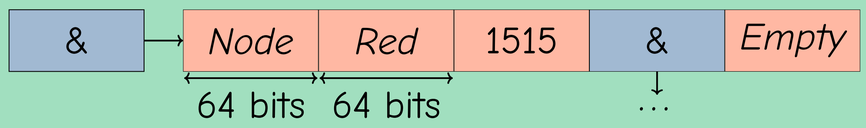
Memory representations

- Pattern matching compilation is **representation-dependent**.
- Current compilation approaches mandate a fixed representation.
- HPC requires efficient, customizable memory representations.

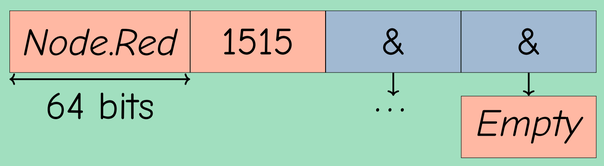
Internal representation of red-black trees ($\text{Repr}^\bullet_{\text{RBT}}$)

Example for *Node*(*Red*, 1515, & … , &*Empty*)

OCaml representation: uniform and composable but inefficient.



Rust representation: more compact, less indirection but less uniform.



Research Goals

- Specify efficient memory representations for ADTs.
- Compile and optimize pattern matching for such representations.

Contributions

- Formalization of memory representations: ($\text{Repr}^\bullet$, $\text{Knit}^\bullet$, $\text{Frog}^\bullet$).
- Toy and real-world representations: OCaml, Scala, "packed"…
- A parametrized algorithm for pattern matching compilation.
- A representation **validity criterion** ensuring **Knit&Frog** correctness.

Implementation: *ribbit*

2.5k LoC in OCaml.

- Input pattern language → evaluation, **compilation**.
- Compiles to **decision trees** (switches on memory values).
- Defining a new representation typically takes 50 LoC.

Representation-*parametrized* compilation approach: Knit&Frog

Based on state-of-the-art pattern matching compilation.

Pattern matching example for type

$\tau_0 = None + Some(A + B + C(\text{i32}))$

$$m = match \left\{ \begin{array}{ccc} (None|Some(A)) & \mapsto & 0 \\ Some(B) & \mapsto & 1 \\ Some(C(n)) & \mapsto & 2 + n \end{array} \right\}$$

To compile this matching (cf. steps on the right), we use:

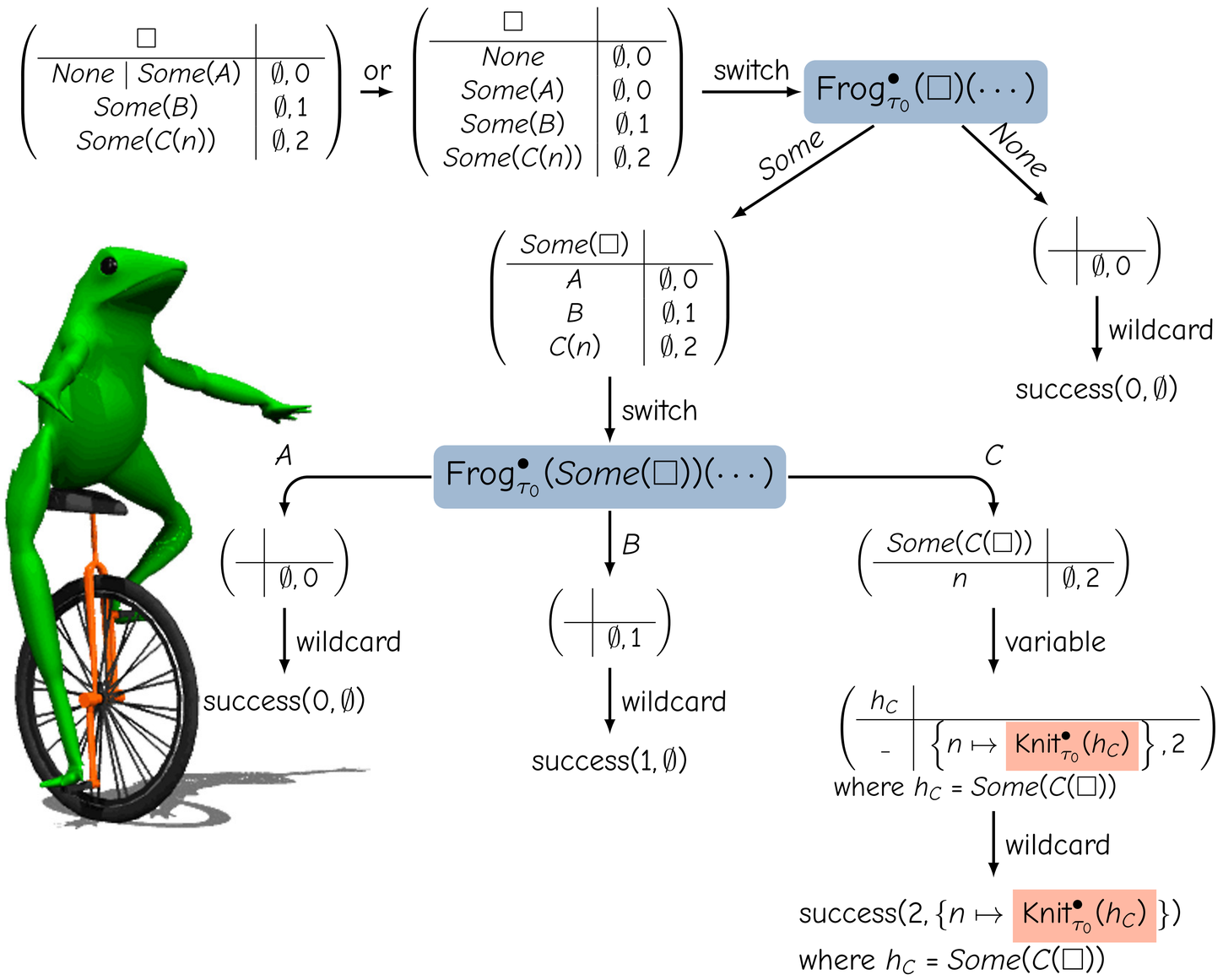
- $\text{Knit}^\bullet_\tau(h)$ emits code that rebuilds the representation of a subterm from its position in the parent term.
- $\text{Frog}^\bullet_\tau(h)(K_0 \mapsto T_0, \dots, K_{n-1} \mapsto T_{n-1})$ emits code that destructs a memory value and branches to its associated decision tree.

Finally, we obtain the decision tree below.

Produced code for the example

switch(head constructor of v_0){
  case *None* : 0;
  case *Some* :
    switch(head constructor of $Some(\square)[v_0]$){
      case *A* : 0;
      case *B* : 1;
      case *C* : 2 + i32 representation of $Some(C(\square))[v_0]$;
  }}

Steps of our compilation algorithm on the example



http://www.inria.fr/ http://www.ens-lyon.eu/ http://www.ens-lyon.fr/LIP/ http://www.univ-lyon1.fr/ http://www.cnrs.fr/ https://lcis.grenoble-inp.fr/