Staging with quotation
(Metaprogramming 2018)

Jeremy Yallop
“People confuse the familiar for the simple. For new features, people insist on LOUD explicit syntax. For established features, people want terse notation.”

-- Bjarne Stroustrup

— Nada Amin
Why quotation?

**Aim**: optimization (mostly)

**Write**: high-level, general-purpose programs

**Generate**: low-level, specialized programs
Quasiquotation in typed functional languages

Scala

MetaOCaml

Haskell

'(1 + ~x)

.< 1 + .~x >.

[ || 1 + $$x$$ ||]
Quasiquotation in Scala

'(e) ~ (e) Expr[A]

delay evaluating e resume evaluating e quoted expression type

Properties

* typed
  well-typed generator → well-typed generated code

* hygienic
  no inadvertent name capture

* generative
  no way to inspect code

---

A Practical Unification of Multi-stage Programming and Macros
N. Stucki, A. Biboudis, M. Odersky
GPCE (2018)
Application: stream fusion

**Idea:** turn stream fusion techniques into a staged library

```ocaml
let sum = fold (fun z a → ≪.~a + .~z≫) ≪0≫ in
  of_arr ≪arr≫
|> map (fun x → ≪.~x * .~x≫)
|> sum
```

**Result:** write high-level combinators, generate fast imperative loops

```ocaml
let s₁ = ref 0 in let arr₂ = arr in
for i₃ = 0 to Array.length arr₂ -1 do
  let el₄ = arr₂.(i₃) in let t₅ = el₄ * el₄ in
  s₁ := t₅ + !s₁
done; !s₁
```

*Stream Fusion, to Completeness*
O. Kiselyov et al.
POPL (2017)
Application: Fast Fourier Transform

Idea: specialize FFT with abstract interpretation techniques

```haskell
let rec fft dir = function
| [] → 1
| l → let (e, o) = split l in
  let y0 = fft dir e and y1 = fft dir o in
  merge dir y0 y1
```

Result: eliminate float operations; generate optimal C code

```plaintext
... y1_385 = x_382[4]; y2_386 = x_382[5];
y1_387 = y1_383 + y1_385;
... 
```

A methodology for generating verified combinatorial circuits
O. Kiselyov, K. Swadi and W. Taha
EMSOFT (2004)
Application: supercompilation

Idea: apply supercompilation techniques during staging

Result: write a naive string matcher, get optimized KMP code

```ml
kmp "aab"
⇝

let rec kmp_aab = function
  | x::xs → if x = 'a' then kmp_ab xs else kmp_aab xs
and kmp_ab = function
  | x::xs → if x = 'a' then kmp_b xs else kmp_aab xs
and kmp_b = function
  | x::xs → if x = 'b' then true
    else if x = 'a' then kmp_b xs
    else kmp_ab xs
```

Supercompiling with staging
J. Inoue
META (2014)
Interpreters to compilers

Idea: add quotation to an interpreter, obtain a compiler

Tagless final interpreter

```
implicit object Eval
    extends Exp[Id] {
        def add = x => y => x + y
        def int = x => x
        def bool = b => b
        def if_ = b => t => e =>
            if (b) { t () }
            else { e () }
        def lam = f => x => f(x)
        def app = f => x => f(x)
    }
```

Tagless final compiler

```
implicit object Compile
    extends Exp[Expr] {
        def add = x => y => '(~x + ~y)
        def int = x => x.toExpr
        def bool = b => b.toExpr
        def if_ = b => t => e =>
            '{if (~b) { ~(t ()) }
            else { ~(e ()) }}
        def lam = f => '{x => f('(x))}
        def app = f => x => '((~f)(~x))
    }
```

Example:

```
lam(f => lam(x => app(f)(app(add(int(3))(app(f)(x))))))
⇝ '{f => x => f (3 + f(x))}
```
Arithmetic optimization

Problem
Naive staging $\leadsto$ inefficient code

Solution
Staging-aware operations

$2 + '(x) + 3 + 4$
$\leadsto$ (reduce)

$2 + '(x) + 7$
$\leadsto$ (residualize)

$(2 + x + 7)$

Partially static data as free extension of algebras
J. Yallop, T. von Glehn, O. Kammar
ICFP (2018)
Staged generic programming

Idea: write high-level data-generic transformations

let rec listify {T:TYPEABLE} p {D:DATA} x =
    mkQ [] (single p) x @ concat (gmapQ (listify p) x)

Instantiate for particular types, producing efficient type-specific code

listify evenp (l : (int * string option) list)

\(\leadsto\)

\[\begin{align*}
\ll & \text{let rec list } l = \text{match } l \text{ with} \\
& | [] \rightarrow [] \\
& | h::t \rightarrow \text{let } (x,y) = h \text{ in} \\
& \quad \text{if } \text{evenp } x \text{ then } x :: \text{list } t \text{ else } \text{list } t \text{ in} \\
& \quad \text{fun } l \rightarrow \text{list } l \gg
\end{align*}\]

Staged generic programming
J. Yallop
ICFP (2017)