Klister:
type inference for type-driven macros

presented by Samuel Gélineau
at the COMEPLS Seminar on 2024-09-17

Klister

1. The goal
2. Straightforward but incorrect approach
3. Working but non-confluent approach
4. The solution

Klister

> 1. The goal
  2. Straightforward but incorrect approach
  3. Working but non-confluent approach
  4. The solution

Klister

Klister

```
#lang "klister.kl"
-- Type-inference

-- const : forall a b. a -> b -> a
(define const
  (lambda (r i)
    r))


(example
  const
)
(example
  (const "hello" 42)
)
```

```
#lang "klister.kl"
-- Type-inference

-- const : forall a. a -> Int -> a
(define const
  (lambda (r i)
    (let [_x (+ i 1)]
      r)))

(example
  const
)
(example
  (const "hello" 42)
)
```

Klister

Klister

```
#lang "klister.kl"
--       Late-typed code generation

-- (const* 3 "hello")
-- =>
-- (const (const (const "hello")))
(define-macro (const* n r)
  (case-integer n
    [zero       (pure r)]
    [(succ n-1) (pure `(const (const* ,n-1 ,r)))]))

(example
  (const* 0 "hello")  -- "hello"
)
(example
  (const* 1 "hello")  -- (const "hello")
)
(example
  (const* 2 "hello")  -- (const (const "hello"))
)
```

```
#lang "klister.kl"
--       Late-typed code generation

-- (const* 3 "hello")
-- =>
-- (const (const (const "hello")))
(define-macro (const* n r)
  (case-integer n
    [zero       (pure r)]
    [(succ n-1) (pure `(const (const* ,n-1 ,r)))]))

(example
  (const* 0 "hello")  -- "hello"
)                     --   : String
(example
  (const* 1 "hello")  -- (const "hello")
)                     --   : (-> Int String)
(example
  (const* 2 "hello")  -- (const (const "hello"))
)                     --   : (-> Int Int String)
```

```
#lang Beluga, Moebius
--       Late-typed code generation
-- vs [ Early-typed code generation ]

power : int -> [ x:int |- int ]
power n =
  if n = 0
  then box(x. 1)
  else let box(x. X_TO_THE_N_MINUS_ONE) = power (n - 1)
    in box(x.
         (X_TO_THE_N_MINUS_ONE with x) * x
       )
```

```
#lang Beluga, Moebius
--       Late-typed code generation
-- vs [ Early-typed code generation ]

power : int -> [ x:int |- int ]
power n =
  if n = 0
  then box(x. 1)
  else let box(x. X_TO_THE_N_MINUS_ONE) = power (n - 1)
    in box(x.
         (X_TO_THE_N_MINUS_ONE with x) ++ x  -- expected string, got int
       )  -- expected [ x:int |- int ], got [ x:int |- string ]
```

```
#lang "klister.kl"
--     [  Late-typed code generation ]
-- vs   Early-typed code generation

-- const : (-> Int Syntax (Macro Syntax))
(define-macro (power n x)
  (case-integer n
    [zero       (pure `1)]
    [(succ n-1) (pure `(*  ,x (power ,n-1 ,x)))]))

(example
  (power 8 2)
)
```

```
#lang "klister.kl"
--     [  Late-typed code generation ]
-- vs   Early-typed code generation

-- power : (-> Int Syntax (Macro Syntax))
(define-macro (power n x)
  (case-integer n
    [zero       (pure `1)]
    [(succ n-1) (pure `(++ ,x (power ,n-1 ,x)))]))  -- typechecks!

(example
  (power 8 2)  -- expected String, got Int
)
```

Klister

Klister

```
#lang "klister.kl"
-- Type-driven code generation

-- default : (Macro Syntax)
(define-macro (default)
  (do (t <- (expected-type))
      (type-case t
        [Int       (pure `1)]
        [String    (pure `"!")]
        [(-> a b) (type-case a
                    [Int    (pure `(lambda (x) (+ x 1)))]
                    [String (pure `(lambda (x) (++ x "!")))])])))

(example
  (+ 42 (default))          -- (default)
)                           --   : Int
(example                    -- (default)
  (++ "hello" (default))    --   : String
)
(example                    -- (default)
  ((default) 42)            --   : (-> Int ?1)
)
(example                    -- (default)
  ((default) "hello")       --   : (-> String ?2)
)
```

```
#lang "klister.kl"
-- Type-driven code generation

-- default : (Macro Syntax)
(define-macro (default)
  (do (t <- (expected-type))
      (type-case t
        [Int       (pure `1)]
        [String    (pure `"!")]
        [(-> a b) (type-case a
                    [Int    (pure `(lambda (x) (+ x 1)))]
                    [String (pure `(lambda (x) (++ x "!")))])])))

(example
  (+ 42 (default))          -- 1
)                           --   : Int
(example                    -- "hello"
  (++ "hello" (default))    --   : String
)
(example                    -- (lambda (x) (+ x 1))
  ((default) 42)            --   : (-> Int ?1)
)
(example                    -- (lambda (x) (++ x "!"))
  ((default) "hello")       --   : (-> String ?2)
)
```

```
#lang "klister.kl"
-- Type-driven code generation

-- default : (Macro Syntax)
(define-macro (default)
  (do (t <- (expected-type))
      (type-case t
        [Int       (pure `1)]
        [String    (pure `"!")]
        [(-> a b) (type-case a
                    [Int    (pure `(lambda (x) (+ x 1)))]
                    [String (pure `(lambda (x) (++ x "!")))])])))

(example
  (+ 42 (default))          -- 1
)                           --   : Int
(example                    -- "hello"
  (++ "hello" (default))    --   : String
)
(example                    -- (lambda (x) (+ x 1))
  ((default) 42)            --   : (-> Int Int)
)
(example                    -- (lambda (x) (++ x "!"))
  ((default) "hello")       --   : (-> String String)
)
```

Klister

Klister

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const* 1 "hello")
   (default)))            -- (default)

(example
  ((default)              -- (default)
   (const* 0 "hello")))   -- (const* 0 "hello")

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello")
   (default)))            -- (default)

(example
  ((default)              -- (default)
   (const* 0 "hello")))   -- (const* 0 "hello")

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- (default)

(example
  ((default)              -- (default)
   (const* 0 "hello")))   -- (const* 0 "hello")

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- (default) : Int

(example
  ((default)              -- (default)
   (const* 0 "hello")))   -- (const* 0 "hello")

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- 1 : Int

(example
  ((default)              -- (default)
   (const* 0 "hello")))   -- (const* 0 "hello")

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- 1 : Int

(example
  ((default)              -- (default)
   (const* 0 "hello")))   -- "hello"

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- 1 : Int

(example
  ((default)              -- (default)
   (const* 0 "hello")))   -- "hello" : String

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- 1 : Int

(example
  ((default)              -- (default) : (-> String ?1)
   (const* 0 "hello")))   -- "hello" : String

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- 1 : Int

(example
  ((default)              -- (lambda (x) (++ x "!") : (-> String ?1)
   (const* 0 "hello")))   -- "hello" : String

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- 1 : Int

(example
  ((default)              -- (lambda (x) (++ x "!") : (-> String String)
   (const* 0 "hello")))   -- "hello" : String

--(example
--  ((default)            -- (default)
--   (default))           -- (default)
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- 1 : Int

(example
  ((default)              -- (lambda (x) (++ x "!") : (-> String String)
   (const* 0 "hello")))   -- "hello" : String

--(example
--  ((default)            -- (default) : (-> ?1 ?2)
--   (default))           -- (default) : ?1
--)
```

```
#lang "klister.kl"
-- Late-typed code generation + Type-driven code generation

(example
  ((const* 1 "hello")     -- (const "hello") : (-> Int String)
   (default)))            -- 1 : Int

(example
  ((default)              -- (lambda (x) (++ x "!") : (-> String String)
   (const* 0 "hello")))   -- "hello" : String

--(example                -- error: type is ambiguous
--  ((default)            -- (default) : (-> ?1 ?2)
--   (default))           -- (default) : ?1
--)
```

Klister

Klister

Klister

> 1. The goal
  2. Straightforward but incorrect approach
  3. Working but non-confluent approach
  4. The solution

Klister

1. The goal
> 2. Straightforward but incorrect approach
3. Working but non-confluent approach
4. The solution

```
#lang "klister.kl"
-- Compiler phases

--    parser
--       v
--  typechecker
--       v
-- code generator
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--   expander
--       v
--  typechecker
--       v
-- code generator
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    [parser]
--        v
--    expander
--        v
--  typechecker
--        v
-- code generator

(example
  ((const* 2 "hello")
   1
   2)
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--  [expander]
--       v
--  typechecker
--       v
-- code generator

(example
  ((const* 2 "hello")
   1
   2)
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--  [expander]
--       v
--  typechecker
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const* 2 "hello")
   1
   2)
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--  [expander]
--       v
--  typechecker
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1
   2)
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--   expander
--       v
-- [typechecker]
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1
   2)
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--   expander
--       v
-- [typechecker]
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1                   --    : (-> Int Int String)
   2)  -- : String
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--   expander
--       v
--  typechecker
--       v
--[code generator]

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1                   --    : (-> Int Int String)
   2)  -- : String
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--   expander
--       v
--  typechecker
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1                   --    : (-> Int Int String)
   2)  -- : String
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--   expander
--       v
--  typechecker
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1                   --    : (-> Int Int String)
   2)  -- : String
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--   [parser]
--       v
--   expander
--       v
--  typechecker
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1                   --   : (-> Int Int String)
   2)  -- : String
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--  [expander]
--       v
--  typechecker
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1                   --    : (-> Int Int String)
   2)  -- : String
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Straightforward approach: expand the macros in the expander phase

--    parser
--       v
--  [expander]
--       v
--  typechecker
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1                   --   : (-> Int Int String)
   2)  -- : String
)

(example
  (+ 42
     (default))        -- (default)
)
```

```
#lang "klister.kl"
-- Type-driven macros need type information!

--    parser
--       v
--  [expander]
--       v
--  typechecker
--       v
-- code generator

(example
  ((const* 2 "hello")  -- (const (const "hello"))
   1                   --   : (-> Int Int String)
   2)  -- : String
)

(example
  (+ 42
     (default))        -- (default) : ?
)
```

```
#lang "klister.kl"
-- Type-check first?

--    parser               parser
--       v                    v
--   expander           typechecker
--       v                    v
--  typechecker          expander
--       v                    v
-- code generator      code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Type-check first?

--    parser              [parser]
--       v                   v
--   expander           typechecker
--       v                   v
--  typechecker          expander
--       v                   v
-- code generator      code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Type-check first?

--    parser               parser
--       v                    v
--   expander           [typechecker]
--       v                    v
--  typechecker           expander
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Type-check first?

--    parser                parser
--       v                     v
--   expander            [typechecker]
--       v                     v
--  typechecker            expander
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42                  -- + : (-> Int Int Int)
     (default))
)
```

```
#lang "klister.kl"
-- Type-check first?

--    parser                parser
--       v                     v
--   expander             [typechecker]
--       v                     v
--  typechecker             expander
--       v                     v
-- code generator        code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42                  -- + : (-> Int Int Int)
     (default))          -- (default) : Int
)
```

```
#lang "klister.kl"
-- Type-check first?

--    parser                parser
--       v                     v
--   expander             typechecker
--       v                     v
--  typechecker           [expander]
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42                  -- + : (-> Int Int Int)
     (default))          -- (default) : Int
)
```

```
#lang "klister.kl"
-- Type-check first?

--    parser                parser
--       v                     v
--   expander             typechecker
--       v                     v
--  typechecker           [expander]
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42                  -- + : (-> Int Int Int)
     (default))          -- 1 : Int
)
```

```
#lang "klister.kl"
-- Late-typed macros must be type-check after!

--    parser                parser
--       v                     v
--   expander             typechecker
--       v                     v
--  typechecker           [expander]
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42                  -- + : (-> Int Int Int)
     (default))          -- "!" : Int
)
```

```
#lang "klister.kl"
-- Late-typed macros must be type-check after!

--    parser                parser
--       v                     v
--   expander             typechecker
--       v                     v
--  typechecker            expander
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42                  -- + : (-> Int Int Int)
     (default))          -- "!" : Int
)
```

```
#lang "klister.kl"
-- Late-typed macros must be type-check after!

--    parser               [parser]
--       v                    v
--   expander            typechecker
--       v                    v
--  typechecker           expander
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42                -- + : (-> Int Int Int)
     (default))        -- "!" : Int
)
```

```
#lang "klister.kl"
-- Late-typed macros must be type-check after!

--    parser                parser
--       v                     v
--   expander            [typechecker]
--       v                     v
--  typechecker            expander
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42                 -- + : (-> Int Int Int)
     (default))         -- "!" : Int
)
```

```
#lang "klister.kl"
-- Late-typed macros must be type-check after!

--    parser                parser
--       v                     v
--   expander            [typechecker]
--       v                     v
--  typechecker            expander
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1                     -- 1 : Int
   2)                    -- 1 : Int
)

(example
  (+ 42                  -- + : (-> Int Int Int)
     (default))          -- "!" : Int
)
```

```
#lang "klister.kl"
-- Late-typed macros must be type-check after!

--    parser                parser
--       v                     v
--   expander            [typechecker]
--       v                     v
--  typechecker            expander
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- (const* 2 "hello") : (-> Int Int ?1)
   1                   -- 1 : Int
   2)                  -- 1 : Int
)

(example
  (+ 42                -- + : (-> Int Int Int)
     (default))        -- "!" : Int
)
```

```
#lang "klister.kl"
-- The type-checker needs the final code to infer the full type!

--    parser                parser
--       v                     v
--   expander            [typechecker]
--       v                     v
--  typechecker            expander
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ? : (-> Int Int ?1)
   1                   -- 1 : Int
   2)                  -- 1 : Int
)

(example
  (+ 42                -- + : (-> Int Int Int)
     (default))        -- "!" : Int
)
```

Klister

Klister

Klister

1. The goal
> 2. Straightforward but incorrect approach
3. Working but non-confluent approach
4. The solution

Klister

1. The goal
2. Straightforward but incorrect approach
> 3. Working but non-confluent approach
4. The solution

Klister

Klister

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser              parser                  parser
--       v                   v                       v
--   expander           typechecker          expander + typechecker
--       v                   v                       v
--  typechecker           expander               code generator
--       v                   v
-- code generator      code generator
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander            typechecker          expander + typechecker
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                  [parser]
--       v                    v                       v
--   expander            typechecker          expander + typechecker
--       v                    v                       v
--  typechecker           expander               code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander           typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                    parser
--       v                     v                         v
--   expander             typechecker          [expander + typechecker]
--       v                     v                         v
--  typechecker            expander                code generator
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ( _
   1                   --   _
   2)                  --   _ )
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                    parser
--      v                     v                         v
--   expander             typechecker          [expander + typechecker]
--      v                     v                         v
--  typechecker            expander                code generator
--      v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ((_ : (-> ?1 ?2 ?3))
   1                   --  (_ : ?1)
   2)                  --  (_ : ?2)) : ?3
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander           typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")  --   _ : (-> ?1 ?2 ?3)
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander            typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")  --  (const* 2 "hello") : (-> ?1 ?2 ?3)
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander            typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")  --  (const (const "hello")) : (-> ?1 ?2 ?3)
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander            typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")  --  (const (const "hello")) : (-> Int Int String)
   1
   2)
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                    parser
--       v                     v                         v
--   expander             typechecker          [expander + typechecker]
--       v                     v                         v
--  typechecker            expander                code generator
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ((const (const "hello")) : (-> Int Int String)
   1                   --  (_ : ?1)
   2)                  --  (_ : ?2)) : ?3
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander            typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ((const (const "hello")) : (-> Int Int String)
   1                   --  (_ : Int)
   2)                  --  (_ : Int)) : String
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander            typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ((const (const "hello")) : (-> Int Int String)
   1                   --  (1 : Int)
   2)                  --  (2 : Int)) : String
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                    parser
--       v                     v                         v
--   expander             typechecker           expander + typechecker
--       v                     v                         v
--  typechecker            expander                [code generator]
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")   -- ((const (const "hello")) : (-> Int Int String)
   1                    --  (1 : Int)
   2)                   --  (2 : Int)) : String
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                    parser
--       v                     v                         v
--   expander             typechecker           expander + typechecker
--       v                     v                         v
--  typechecker            expander                code generator
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ((const (const "hello")) : (-> Int Int String)
   1                   --  (1 : Int)
   2)                  --  (2 : Int)) : String
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                    [parser]
--       v                     v                         v
--   expander             typechecker           expander + typechecker
--       v                     v                         v
--  typechecker            expander                code generator
--       v                     v
-- code generator        code generator

(example
  ((const* 2 "hello")   -- ((const (const "hello")) : (-> Int Int String)
   1                    --  (1 : Int)
   2)                   --  (2 : Int)) : String
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander            typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ((const (const "hello")) : (-> Int Int String)
   1                   --  (1 : Int)
   2)                  --  (2 : Int)) : String
)

(example
  (+ 42
     (default))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander            typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ((const (const "hello")) : (-> Int Int String)
   1                   --  (1 : Int)
   2)                  --  (2 : Int)) : String
)

(example               -- ( _
  (+ 42                --   _
     (default))        --   _ )
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                      parser
--       v                     v                           v
--   expander             typechecker           [expander + typechecker]
--       v                     v                           v
--  typechecker            expander                 code generator
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")   -- ((const (const "hello")) : (-> Int Int String)
   1                    --  (1 : Int)
   2)                   --  (2 : Int)) : String
)

(example                -- ((+ : (-> Int Int Int))
  (+ 42                 --  (42 : Int)
     (default))         --  ((default) : Int))
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser               parser                    parser
--       v                    v                         v
--   expander            typechecker          [expander + typechecker]
--       v                    v                         v
--  typechecker           expander                code generator
--       v                    v
-- code generator       code generator

(example
  ((const* 2 "hello")  -- ((const (const "hello")) : (-> Int Int String)
   1                   --  (1 : Int)
   2)                  --  (2 : Int)) : String
)

(example
  (+ 42
     (default))        --    (default) : Int
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                    parser
--       v                     v                         v
--   expander             typechecker          [expander + typechecker]
--       v                     v                         v
--  typechecker            expander                code generator
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")   -- ((const (const "hello")) : (-> Int Int String)
   1                    --  (1 : Int)
   2)                   --  (2 : Int)) : String
)

(example
  (+ 42
     (default))         --   1 : Int
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                      parser
--       v                     v                           v
--   expander             typechecker           [expander + typechecker]
--       v                     v                           v
--  typechecker            expander                 code generator
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")   -- ((const (const "hello")) : (-> Int Int String)
   1                    --  (1 : Int)
   2)                   --  (2 : Int)) : String
)

(example
  (+ 42
     (default))         --   "!" : Int
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                    parser
--       v                     v                         v
--   expander             typechecker          [expander + typechecker]
--       v                     v                         v
--  typechecker            expander                code generator
--       v                     v
-- code generator       code generator

(example
  ((const* 2 "hello")   -- ((const (const "hello")) : (-> Int Int String)
   1                    --  (1 : Int)
   2)                   --  (2 : Int)) : String
)

(example
  (+ 42
     (default))         --   1 : Int
)
```

```
#lang "klister.kl"
-- Interleaving macro-expansion and type-checking

--    parser                parser                      parser
--      v                     v                           v
--   expander             typechecker           [expander + typechecker]
--      v                     v                           v
--  typechecker            expander                 code generator
--      v                     v
-- code generator        code generator

(example
  ((const* 2 "hello")   -- ((const (const "hello")) : (-> Int Int String)
   1                    --  (1 : Int)
   2)                   --  (2 : Int)) : String
)

(example                -- ((+ : (-> Int Int Int))
  (+ 42                 --  (42 : Int)
     (default))         --  (1 : Int)) : Int
)
```

Klister

Klister

```
#lang "klister.kl"
-- Fragile

(example                -- ((+ : (-> Int Int Int))
  (+ 42                 --  (42 : Int)
     (default))         --  (1 : Int)) : Int
)
```

```
#lang "klister.kl"
-- Fragile

(example                -- ((+ : (-> Int Int Int))
  (+ 42                 --  (42 : Int)
     (default))         --  (1 : Int)) : Int
)

(example
  (let [x (default)]
    (+ 42 x))
)
```

```
#lang "klister.kl"
-- Fragile

(example                -- ((+ : (-> Int Int Int))
  (+ 42                 --  (42 : Int)
     (default))         --  (1 : Int)) : Int
)

(example
  (let [x (default)]    -- (default) : ?1
    (+ 42 x))
)
```

```
#lang "klister.kl"
-- Fragile

(example                -- ((+ : (-> Int Int Int))
  (+ 42                 --  (42 : Int)
     (default))         --  (1 : Int)) : Int
)

(example
  (let [x (default)]    -- (default) : ?1
    (+ 42 x))
)

--(example               -- error: type is ambiguous
--  ((default)          -- (default) : (-> ?1 ?2)
--   (default))         -- (default) : ?1
--)
```

```
#lang "klister.kl"
-- Fragile

(example                -- ((+ : (-> Int Int Int))
  (+ 42                 --  (42 : Int)
     (default))         --  (1 : Int)) : Int
)

(example                -- error: type is ambiguous
  (let [x (default)]    -- (default) : ?1
    (+ 42 x))
)

--(example               -- error: type is ambiguous
--  ((default)          -- (default) : (-> ?1 ?2)
--   (default))         -- (default) : ?1
--)
```

Klister

Klister

```
#lang "klister.kl"
-- Non-confluence

                        -- function position,       argument position,
                        -- then argument position   then function position

(example                --
  ((const* 1 "hello")   --
   (default))           --
)
```

```
#lang "klister.kl"
-- Non-confluence

                        -- function position,       argument position,
                        -- then argument position   then function position

(example                --
  ((const* 1 "hello")   -- (const* 1 "hello")
   (default))           --                         (default)
)
```

```
#lang "klister.kl"
-- Non-confluence

                        -- function position,       argument position,
                        -- then argument position   then function position

(example                --
  ((const* 1 "hello")   -- (const "hello")
   (default))           --                          (default)
)
```

```
#lang "klister.kl"
-- Non-confluence

                        -- function position,       argument position,
                        -- then argument position   then function position

(example                --
  ((const* 1 "hello")   -- (const "hello")
   (default))           -- (default) : Int          (default)
)
```

```
#lang "klister.kl"
-- Non-confluence

                        -- function position,       argument position,
                        -- then argument position   then function position

(example                --
  ((const* 1 "hello")   -- (const "hello")
   (default))           -- 1 : Int                   (default)
)
```

```
#lang "klister.kl"
-- Non-confluence

                          -- function position,       argument position,
                          -- then argument position   then function position

(example                  --                          error: type is ambiguous
  ((const* 1 "hello")     -- (const "hello")
   (default))             -- 1 : Int                  (default) : ?1
)
```

```
#lang "klister.kl"
-- Non-confluence

                          -- function position,       argument position,
                          -- then argument position   then function position

(example                  --                          error: type is ambiguous
  ((const* 1 "hello")     -- (const "hello")
   (default))             -- 1 : Int                  (default) : ?1
)
(example                  -- error: type is ambiguous
  ((default)              -- (default) : (-> ?1 ?2)   (lambda (x) (+ x 1))
   (const* 0 "hello"))    --                          "hello"
)
```

Klister

Klister

Klister

1. The goal
2. Straightforward but incorrect approach
> 3. Working but non-confluent approach
4. The solution

Klister

1. The goal
2. Straightforward but incorrect approach
3. Working but non-confluent approach
> 4. The solution

Klister

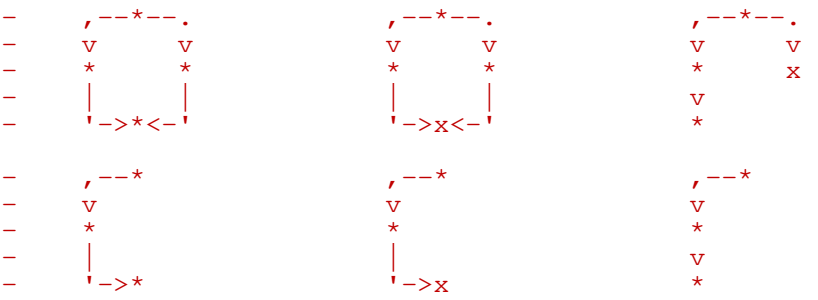
Klister

```
#lang "klister.kl"
-- Removing transitions

((const* 1 "hello")
 (const* 0 "hello"))

--     ,--*--.
--     v     v
--     *     *
--     |     |
--     '->*<-'
```

```
#lang "klister.kl"
-- Removing transitions

((const* 1 "hello")     ((default)
 (const* 0 "hello"))     (default))
```

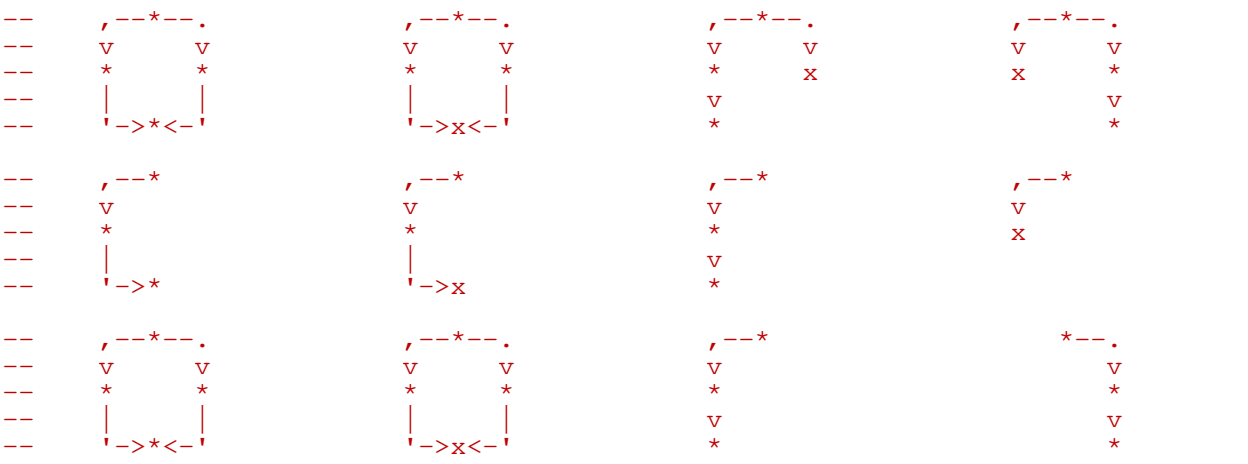
```
#lang "klister.kl"
-- Removing transitions

((const* 1 "hello")     ((default)     ((const* 1 "hello")
 (const* 0 "hello"))     (default))     (default))
```

```
#lang "klister.kl"
-- Removing transitions

((const* 1 "hello")     ((default)     ((const* 1 "hello")
 (const* 0 "hello"))     (default))     (default))
```



```
#lang "klister.kl"
-- Removing transitions

((const* 1 "hello")     ((default)     ((const* 1 "hello")   ((default)
 (const* 0 "hello"))     (default))     (default))            (const* 0 "hello")

--     ,--*--.           ,--*--.           ,--*--.           ,--*--.
--     v     v           v     v           v     v           v     v
--     *     *           *     *           *     x           x     *
--     |     |           |     |           v                       v
--     '->*<-'           '->x<-'           *                       *

--     ,--*              ,--*              ,--*              ,--*
--     v                 v                 v                 v
--     *                 *                 *                 x
--     |                 |                 v
--     '->*              '->x              *
```

```
#lang "klister.kl"
-- Removing transitions

((const* 1 "hello")     ((default)     ((const* 1 "hello")   ((default)
 (const* 0 "hello"))     (default))     (default))            (const* 0 "hello")
```



Klister

Klister

```
#lang "klister.kl"
-- Stuck macros

                        -- function position,       argument position,
                        -- then argument position   then function position

(example
  ((const* 1 "hello")   --                          error: type is ambiguous
   (default))           --                          (default) : ?1
)
(example                -- error: type is ambiguous
  ((default)            -- (default) : (-> ?1 ?2)
   (const* 0 "hello"))
)
```

```
#lang "klister.kl"
-- Stuck macros



(example                  -- (type-case ?1
  ((const* 1 "hello")     --   [Int          e1]  ---/-->  error: type is ambiguous
   (default))             --   [(-> Int t2) e2])
)
(example                  -- (type-case (-> ?1 ?2)
  ((default)              --   [Int          e1]  ---/-->  error: type is ambiguous
   (const* 0 "hello"))    --   [(-> Int t2) e2])
)
```

Klister

Klister

```
#lang "klister.kl"
-- Task queue

(example
  ((const* 1 "hello")
   (default))
)
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const* 1 "hello")
   (default))
)


-- TASK QUEUE
--
--   * expand + typecheck ((const* 1 "hello")
--                         (default))
```

```
#lang "klister.kl"
-- Task queue

(example
  !1
)



-- TASK QUEUE
--
--   * expand + typecheck ((const* 1 "hello") to !1 and ?1
--                         (default))
```

```
#lang "klister.kl"
-- Task queue

(example
  !1
)



-- TASK QUEUE
--
-- > * expand + typecheck ((const* 1 "hello") to !1 and ?1
--                         (default))
```

```
#lang "klister.kl"
-- Task queue

(example
  (!3
   !2)
)


-- TASK QUEUE
--
-- > * expand + typecheck ((const* 1 "hello") to (!3 !2) and ?1
--                         (default))
--   * expand + typecheck (const* 1 "hello") to !3 and (-> ?2 ?1)
--   * expand + typecheck (default) to !2 and ?2
```

```
#lang "klister.kl"
-- Task queue

(example
  (!3
   !2)
)


-- TASK QUEUE
--
--   * expand + typecheck (const* 1 "hello") to !3 and (-> ?2 ?1)
--   * expand + typecheck (default) to !2 and ?2
```

```
#lang "klister.kl"
-- Task queue

(example
  (!3
   !2)
)


-- TASK QUEUE
--
--   * expand + typecheck (const* 1 "hello") to !3 and (-> ?2 ?1)
-- > * expand + typecheck (default) to !2 and ?2
```

```
#lang "klister.kl"
-- Task queue

(example
  (!3
   !2)
)


-- TASK QUEUE
--
--   * expand + typecheck (const* 1 "hello") to !3 and (-> ?2 ?1)
-- > * evaluate (do (t <- (expected-type))
--                  (type-case t
--                    [Int  (pure `1)]
--                    ...))                to !2 and ?2
```

```
#lang "klister.kl"
-- Task queue

(example
  (!3
   !2)
)


-- TASK QUEUE
--
--   * expand + typecheck (const* 1 "hello") to !3 and (-> ?2 ?1)
-- > * evaluate (do
--                  (type-case ?2
--                    [Int  (pure `1)]
--                    ...))                to !2 and ?2
```

```
#lang "klister.kl"
-- Task queue

(example
  (!3
   !2)
)


-- TASK QUEUE
--
--   * expand + typecheck (const* 1 "hello") to !3 and (-> ?2 ?1)
-- > * evaluate (type-case ?2
--                [Int  (pure `1)]
--                ...)              to !2 and ?2
```

```
#lang "klister.kl"
-- Task queue

(example
  (!3
   !2)
)


-- TASK QUEUE
--
--   * expand + typecheck (const* 1 "hello") to !3 and (-> ?2 ?1)
-- > * evaluate (type-case ?2
--                [Int  (pure `1)]
--                ...)              to !2 and ?2  [STUCK ON ?2]
```

```
#lang "klister.kl"
-- Task queue

(example
  (!3
   !2)
)


-- TASK QUEUE
--
-- > * expand + typecheck (const* 1 "hello") to !3 and (-> ?2 ?1)
--   * evaluate (type-case ?2
--                [Int  (pure `1)]
--                ...)              to !2 and ?2  [STUCK ON ?2]
```

```
#lang "klister.kl"
-- Task queue

(example
  (!3
   !2)
)


-- TASK QUEUE
--
-- > * evaluate (case-integer 1 ...) to !3 and (-> ?2 ?1)
--   * evaluate (type-case ?2
--                [Int  (pure `1)]
--                ...)              to !2 and ?2  [STUCK ON ?2]
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   !2)
)


-- TASK QUEUE
--
-- > * expand + typecheck (const "hello") to !3 and (-> ?2 ?1)
--   * evaluate (type-case ?2
--                [Int  (pure `1)]
--                ...)              to !2 and ?2  [STUCK ON ?2]
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   !2)
)


-- TASK QUEUE
--
-- > * expand + typecheck (const "hello") to !3 and (-> Int String)
--   * evaluate (type-case ?2
--                [Int  (pure `1)]
--                ...)              to !2 and ?2  [STUCK ON ?2]
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   !2)
)


-- TASK QUEUE
--
-- > * expand + typecheck (const "hello") to !3 and (-> Int String)
--   * evaluate (type-case Int
--                [Int  (pure `1)]
--                ...)              to !2 and Int [STUCK ON Int]
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   !2)
)


-- TASK QUEUE
--
--   * evaluate (type-case Int
--                [Int  (pure `1)]
--                ...)              to !2 and Int [STUCK ON Int]
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   !2)
)


-- TASK QUEUE
--
--   * evaluate (type-case Int
--                [Int  (pure `1)]
--                ...)              to !2 and Int
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   !2)
)


-- TASK QUEUE
--
-- > * evaluate (type-case Int
--                [Int  (pure `1)]
--                ...)              to !2 and Int
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   !2)
)


-- TASK QUEUE
--
-- > * evaluate (pure `1) to !2 and Int
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   !2)
)


-- TASK QUEUE
--
-- > * expand + typecheck 1 to !2 and Int
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   1)
)


-- TASK QUEUE
--
-- > * expand + typecheck 1 to !2 and Int
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   "!")
)


-- TASK QUEUE
--
-- > * expand + typecheck "!" to !2 and Int
```

```
#lang "klister.kl"
-- Task queue

(example
  ((const "hello")
   1)
)


-- TASK QUEUE
```

```
#lang "klister.kl"
-- Task queue

(example
  ((default)
   (default))
)


-- TASK QUEUE
--
--   * [STUCK ON ?1]
--   * [STUCK ON ?2]
```

Klister:
type inference for type-driven macros

More information: https://github.com/gelisam/klister

* Compose NYC 2019 talk
* TyDe 2020 extended abstract and talk
* Haskell implementation
* Documentation and examples

Future work:
* inference rules, prove confluence
* type classes via a monotonic map
* repl (CEK machine, cross-phase stack trace)
* local-expand (Edinburgh LCF, expand to datatype)

QUESTIONS?