

## Metaprogramming

The art of generating, manipulating, and analyzing code

(Used for Compiler, Program Analyzer, Program generator, etc.)

Direct Programming

Program

Result

Metaprogramming

Meta-program

Object program

Result

## Metaprogramming with Box/Let-box

`box(2 + 3)`

A code fragment for `2 + 3`

`let box(C) = box(2 + 3) in C`

Execution of code `2 + 3` to get `5`

```
untypedNth n = if n <= 0 then box(head xs)
              else let box(C) = untypedNth (n - 1) in
                  box(let xs = tail xs in C)
```

Untyped metaprogram to get the  $n$ -th item in a list

If we evaluate `untypedNth 2`, we get

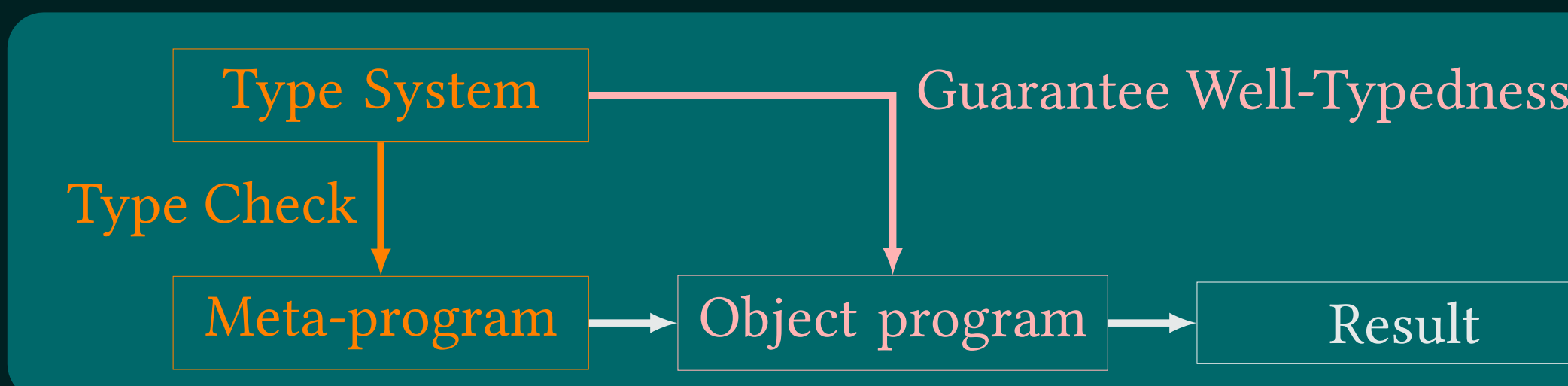
```
box(let xs = tail xs in
    let xs = tail xs in
    head xs)
```

## Typed Metaprogramming

What if

- There is no variable  $xs$  when executing the code from `untypedNth 2`?
- The variable  $xs$  is not a list when executing the code from `untypedNth 2`?

We want to ensure that generated code will be well-typed *before* executing a meta-program.



## Modal Logic

Well-typed Closed  
Code Fragment

Well-typed under  
*Any* Environment

Necessary Truth (in S4)

True under *Any* World

$\Box A$

A type for code generating  $A$ -typed program

`box(M)` has type  $\Box A$

$M$  has type  $A$  but does not use  
any environment-specific information

```
intNth : Int → □ (Int list → Int)
intNth n = if n <= 0 then box(fun xs => head Int xs)
          else let box(C) = intNth (n - 1) in
              box(fun xs => C (tail Int xs))
```

Typed metaprogram to get the  $n$ -th item in an int list

If we execute `intNth 2`, we get

```
box(fun xs =>
  (fun xs =>
    (fun xs => head xs)
    (tail xs))
  (tail xs))
```

## Problem 1 — Well-typed Open Code Fragments

`untypedNth 2` has a free variable `xs`, but we cannot type check this as-is with Simple S4. Thus, `intNth 2` has some redundant anonymous functions and function calls.

## Problem 2 — Polymorphism

`intNth 2` works only for `Int` lists. The following approach to polymorphism does not work:

```
badPolyNth : ('a : Type) → Int → □ ('a list → 'a)
badPolyNth 'a n = if n <= 0 then box(fun xs => head 'a xs)
                  else let box(C) = badPolyNth 'a (n - 1) in
                      box(fun xs => C (tail 'a xs))
```

Ill-typed polymorphic metaprogram to get the  $n$ -th item in a list

because code `fun xs => head 'a xs` depends on `'a` and thus is not closed.

## Problem 3 — Pattern Matching on Code Fragments

```
nthToOneBefore : □ (Int list → Int) → □ (Int list → Int)
nthToOneBefore nthCode = case nthCode of
| box(fun xs => head xs)      → box(fun xs => head xs)
| box(fun xs => x (tail xs)) → box(x)
```

Metaprogram analyzing a code fragment from `nth` to get code for the  $(n - 1)$ -th item

What would be the type of `x`?

## Contextual Modality: Generalization of Necessity

$[\Gamma \vdash A]$

A type for code for  $A$ -typed program  
that is valid under  $\Gamma$

`box( $\Gamma$  . M)` has type  $[\Gamma \vdash A]$

$M$  has type  $A$  but does not use  
any environment-specific information other than from  $\Gamma$

$M$  with  $\sigma$

Fill the information from  $\Gamma$  in  $M$  with  $\sigma$

```
openIntNth : Int → [xs : Int list ⊢ Int]
openIntNth n = if n <= 0 then
  box(xs . head Int xs)
else
  let box(xs . C) = openIntNth (n - 1) in
  box(xs . C with (tail Int xs))
```

Metaprogram generating an open code fragment for the  $n$ -th item access

If we execute `openIntNth 2`, we get

```
box(xs . head (tail (tail xs)))
```

## Levels: Necessity beyond Necessity

$(\Gamma \vdash^1 A)$

Level 1 template depends on  $\Gamma$  of normal types

$(\Gamma \vdash^n A)$

Level  $n$  template depends on  $\Gamma$   
of normal types and level  $\leq n - 1$  templates

$[\Gamma \vdash^n A]$

$\Gamma$  may contain normal types and  
level  $\leq n - 1$  templates

```
polyNth : ('a : (⊢2 Type)) → Int → [xs : 'a list ⊢1 'a]
polyNth 'a n = if n <= 0 then
  box(xs .1 head 'a xs)
else
  let box(xs .1 C) = polyNth 'a (n - 1) in
  box(xs .1 C (tail 'a xs))
```

Polymorphic metaprogram to get the  $n$ -th item in a list

```
nthToOneBefore : ('a : (⊢2 Type)) → [xs : 'a list ⊢1 'a] → [xs : 'a list ⊢1 'a]
nthToOneBefore nthCode = case nthCode of
| box(xs .1 head xs) → box(xs .1 head xs)
| box(xs .1 head (tail x)) → box(xs .1 head x)
```

Ill-typed metaprogram to get the  $n$ -th item in a list

where the type of `x` is `(xs : 'a list ⊢2 'a list)`

## References

Pfenning&Davies/Contextual Type/Moebius