Type System for Metaprogramming

Junyoung "Clare" Jang
https://Ailrun.github.io

Complogic Group
https://complogic.cs.mcgill.ca/

Metaprogramming

The art of generating, manipulating, and analyzing code

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

Direct Programming

Metaprogramming

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

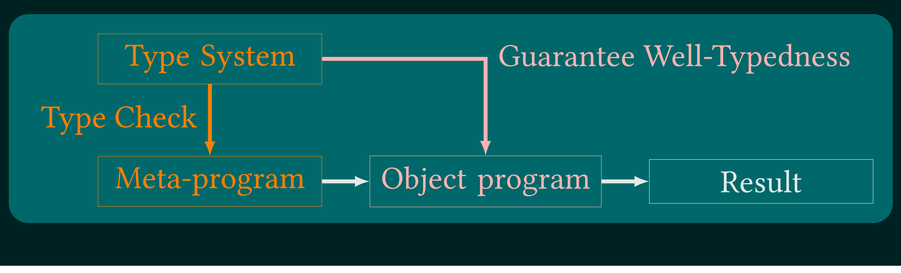
```
box(let xs = tail xs in
    let xs = tail xs in
    head xs)
```

Typed Metaprogramming

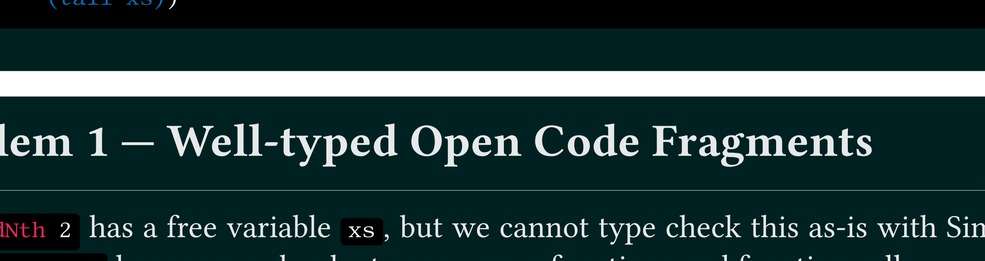
What if

- 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



`□ A` A type for code generating `A`-typed program

`box(M)` has type `□ 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

`[Γ ⊢ A]` A type for code for `A`-typed program that is valid under `Γ`

`box(Γ . M)` has type `[Γ ⊢ A]` `M` has type `A` but does not use any environment-specific information other than from `Γ`

`M with σ` Fill the information from `Γ` in `M` with `σ`

```
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

`(Γ ⊢¹ A)` Level 1 template depends on Γ of normal types

`(Γ ⊢ⁿ A)` Level n template depends on Γ of normal types and level $\leq n-1$ templates

`[Γ ⊢ⁿ A]` `Γ` 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)`

Acknowledgement

This work can be done thanks to the guidance from my supervisor, Prof. Brigitte Pientka, and the help from two other coauthors, Prof. Stefan Monnier at UdeM and MSc. Samuel Gélineau

References

[1] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001.

[2] Junyoung Jang, Samuel Gélineau, Stefan Monnier, and Brigitte Pientka. Mœbius: Metaprogramming using contextual types: The stage where system f can pattern match on itself. *Proc. ACM Program. Lang.*, 6(POPL), jan 2022.

[3] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.