Kripke-Style Contextual Modal Type Theory

YUITO MURASE

THE UNIVERSITY OF TOKYO

Agenda

- Background
- Logic
- Type System
- Future Plan/Related Work

Background: Syntactical Metaprogramming

- Extend the syntax of programming languages
  - Macros in Lisp Family
  - Template Haskell
  - Scala Macros
  - … etc.
- They are not type-safe
  - Well-typed code with syntactic extension can extend to ill-typed code
  - We want type theory for syntactical metaprogramming
    - Especially logical foundation (via the Curry-Howard Isomorphism)

Example: or macro

```
(defmacro or (x y)
  `(if ,x true ,y))
```

- Quasi-quotation: basic construct for code generation
  - Lisp-family, Template Haskell
- Macros are functions from code to code
  - Including open code

```
(or a (is-zero b))
=> (if a true (is-zero b))
```

Example: or macro

```
(defmacro or (x y)
  `(if ,x true ,y))
```

- Quasi-quotation: basic construct for code generation
  - Lisp-family, Template Haskell
- Macros are functions from code to code
  - Including open code

```
(or a (is-zero b))
=> (if a true (is-zero b))
```

Example: or macro

```
(defmacro or (x y)
  `(if ,x true ,y))
```

- Quasi-quotation: basic construct for code generation
  - Lisp-family, Template Haskell
- Macros are functions from code to code
  - Including open code

```
(or a (is-zero b))
=> (if a true (is-zero b))
```

Example: Binding Manipulation

```
(defmacro bind (body)
  `(lambda (x) ,body))
```

- Generate a new binding
- Access to free variables in code

```
(bind (= x x))
 => (lambda (x) (= x x))
```

Background: Modal Type Theory

- Type theory that corresponds to modal logic
  - The Curry-Howard Isomorphism
- $\Box A$
  - Logic : proposition for “A is valid”
  - Type theory : type of “closed code of type A”
- Some formulation for modal logic
  - Dual context formulation
  - Kripke-style formulation

Dual-Context Formulation

- Proposed by Pfenning and Davies[2001]
- Based on the idea of categorical judgment
- Hypothetical judgment have two-levels
  - Object-level and meta-level
  - Syntax includes meta-variables
- Corresponds to S4 modal logic

```
or := λx:□bool. λy:□bool.
        let box u = x in
        let box v = y in
        `(if u then true else v)
```

Kripke-Style Formulation

- Proposed by Martini and Masini(1996), Pfenning and Wong(1995)
- Hypothetical Judgment have context stack
  - Justified by Kripke's multiple-world semantics(1963)
  - Namespace for variables are uniform
- Syntax have quasi-quotaion
- 4 variations: K, T, K4, S4

```
or := λx:□bool. λy:□bool.
       `(if ,x then true else ,v)
```

Contextual Modal Type Theory

- Introduced by Nanevski et al(2007)
- Contextual modality : [Γ]A
  - Logic: A is valid under the context Γ
  - Kripke Semantics: For any next world where Γ holds, A also holds
  - Type: Code with free variables
- Generalization of dual-context modal calculi
- Syntax have meta-variables and explicit substitution

```
bind := λx:[A]bool. let box u = x in
          `(λy:A.u[y])
bind `<x:A>(x==x) → λx:A(x==x)
```

What we want?

- Quasi-quotation → Kripke-style formulation
- The axiom T is not necessary → Kripke-style formulation
  - T corresponds to run-time code evaluation
- Binding manipulation → Contextual modal type
- ⇒ Kripke-style contextual modal type theory

	Dual-context	**Kripke-style**
Modal	Pfenning and Davies[2001]	Martini and Masini[1996] Pfenning and Wong[1995]
Contextual	Nanevski et al[2007]	**HERE!**

Kripke-style Contextual Modal Type Theory

- Another Contextual Modal Type Theory
- Generalization of Kripke-style modal type theory
  - Kripke-style formulation
  - Quasi-quotation
  - Capable of binding-manipulation
  - Four variations (correspondence to K, T, K4, S4)

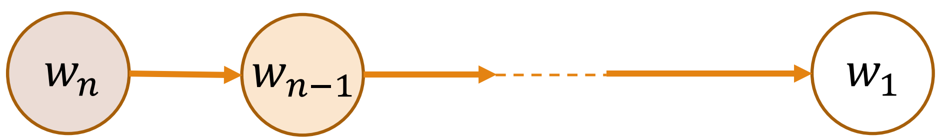
```
bind := λy:[A]bool.
          `<>(λx:A. ,<x>y)
```

Agenda

- Background
- Logic
- Type System
- Future Plan/Related Work

Kripke-style Hypothetical Judgment

- Proposed by [Pfenning and Wong, 1995]
- Contexts form a stack
- Correspondence to Kripke's multiple-world semantics(1963)
  - The stack can be regarded as a sequence of worlds



$$\Gamma_n ; \Gamma_{n-1} ; \ldots ; \Gamma_1 \vdash A$$

Kripke-style Hypothetical Judgment

- Substitution Principle

  If $\Psi; A_1 \ldots A_n; \ldots \vdash T$ and $\Psi; \Gamma \vdash A_i$ holds for all $1 \leq i \leq n$,
  then $\Psi; \Gamma; \ldots \vdash T$.
- Reflexive Principle – assuming reflexivity

  If $\Psi; \Gamma; \Gamma'; \ldots \vdash T$, then $\Psi; \Gamma, \Gamma'; \ldots \vdash T$
- Transitive Principle – assuming transitivity

  If $\Psi; \Gamma; \ldots \vdash T$, then $\Psi; \ldots; \Gamma; \ldots \vdash T$
- Four Variations

Reflexive		✓		✓
Transitive			✓	✓
	K	**T**	**K4**	**S4**

Deduction Rules

$$(\text{hyp})\ \frac{A \in \Gamma}{\boxed{\Psi};\boxed{\Gamma} \vdash A}$$

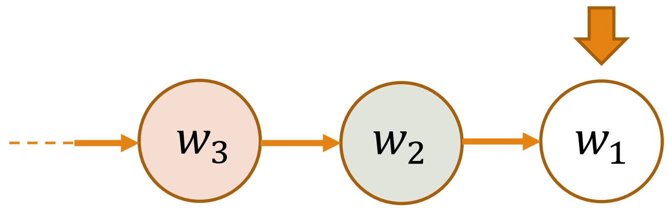
Rest of the Context Stack

Rightmost Context

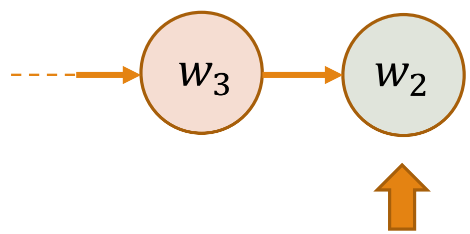
$$(\rightarrow \text{I})\ \frac{\Psi;\Gamma, A \vdash B}{\Psi;\Gamma \vdash A \rightarrow B}$$

$$(\rightarrow \text{E})\ \frac{\Psi;\Gamma \vdash A \rightarrow B \qquad \Psi;\Gamma \vdash A}{\Psi;\Gamma \vdash B}$$

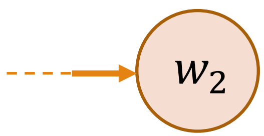
Deduction Rules

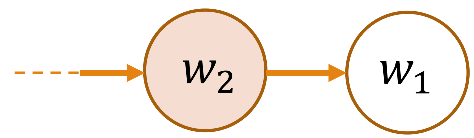


$$([\,]\mathrm{I})\ \frac{\Psi;\Gamma \vdash A}{\Psi \vdash [\Gamma]A}$$

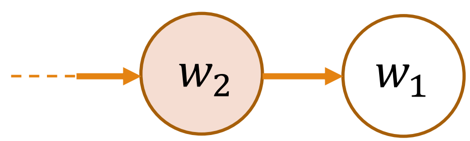


Deduction Rules





$$([\,]\mathrm{E})\ \frac{\Psi \vdash [A_1, \ldots, A_n]B \qquad \Psi; \Gamma \vdash A_i \text{ for } 1 \leq i \leq n}{\Psi; \Gamma \vdash B}$$



Deduction Rules

$$([\,]\mathrm{E}_l)\ \frac{\Psi \vdash [A_1, \ldots, A_n]B \qquad \Psi; \boxed{\Gamma_l; \ldots; \Gamma_1} \vdash A_i \text{ for } 1 \leq i \leq n}{\Psi; \Gamma_l; \ldots; \Gamma_1 \vdash B}$$

$$\text{where } \begin{cases} l = 1 & \text{for K} \\ l = 0, 1 & \text{for T} \\ l \geq 1 & \text{for K4} \\ l \geq 0 & \text{for S4} \end{cases}$$

Examples

1. $\vdash_K [\mathrm{C}](A \to B) \to [C]A \to [C]B$
2. $\vdash_T []A \to A$
3. $\vdash_{K4} [B]A \to [C][B]A$
4. $\vdash_K []A \to [B,C]A$
5. $\vdash_K [B,C]A \to [C,B]A$
6. $\vdash_K [B,B]A \to [B]A$
7. $\vdash_K [B]A \to [C,D]B \to [C,D]A$
8. $\vdash_K [B]A \to [](B \to A)$
9. $\vdash_K [](B \to A) \to [B]A$

Examples

1. $\vdash_K [\mathrm{C}](A \to B) \to [C]A \to [C]B$
2. $\vdash_T []A \to A$
3. $\vdash_{K4} [B]A \to [C][B]A$
4. $\vdash_K []A \to [B, C]A$
5. $\vdash_K [B, C]A \to [C, B]A$
6. $\vdash_K [B, B]A \to [B]A$
7. $\vdash_K [B]A \to [C, D]B \to [C, D]A$
8. $\vdash_K [B]A \to [](B \to A)$
9. $\vdash_K [](B \to A) \to [B]A$

Examples

1. $\vdash_K [\mathrm{C}](A \rightarrow B) \rightarrow [C]A \rightarrow [C]B$
2. $\vdash_T []A \rightarrow A$
3. $\vdash_{K4} [B]A \rightarrow [C][B]A$
4. $\vdash_K []A \rightarrow [B, C]A$
5. $\vdash_K [B, C]A \rightarrow [C, B]A$
6. $\vdash_K [B, B]A \rightarrow [B]A$
7. $\vdash_K [B]A \rightarrow [C, D]B \rightarrow [C, D]A$
8. $\vdash_K [B]A \rightarrow [](B \rightarrow A)$
9. $\vdash_K [](B \rightarrow A) \rightarrow [B]A$

Examples

1. $\vdash_K [\mathrm{C}](A \to B) \to [C]A \to [C]B$
2. $\vdash_T []A \to A$
3. $\vdash_{K4} [B]A \to [C][B]A$
4. $\vdash_K []A \to [B, C]A$
5. $\vdash_K [B, C]A \to [C, B]A$
6. $\vdash_K [B, B]A \to [B]A$
7. $\vdash_K [B]A \to [C, D]B \to [C, D]A$
8. $\vdash_K [B]A \to [](B \to A)$
9. $\vdash_K [](B \to A) \to [B]A$

Agenda

- Background
- Logic
- Type System
- Future Plan/Related Work

Kripke-style Contextual Modal Type Theory

- Correspond to KCML under the Curry-Howard Isomorphism
  - Proposition ⇔ Type
  - Derivation Tree ⇔ Program

Contextual Modal Type

$$
\begin{array}{llrl}
\textbf{Types} & S, T, U & ::= & \tau \mid S \to T \mid \boxed{[S_1, \ldots, S_n]T} \\
\textbf{Terms} & M, N, L & ::= & x \mid \lambda x{:}T.M \mid MN \\
 & & \mid & \boxed{`\langle x_1{:}T_1, \ldots, x_n{:}T_n \rangle M} \mid \boxed{,_l \langle N_1, \ldots, N_n \rangle M} \\
\textbf{Context} & \Gamma, \Delta & ::= & \cdot \mid \Gamma, x{:}T \\
\textbf{Context Stack} & \Psi & ::= & \cdot \mid \Psi; \Gamma \\
\textbf{Judgment} & J & ::= & \Psi \vdash_X M{:}T \text{ (for } X \in \{K, K4, T, S4\}) \\
\end{array}
$$

unquotation

quotation

Typing Rules

$$(\text{Var})\ \frac{x : T \in \Gamma}{\Psi;\Gamma \vdash x : T}$$

$$(\text{Abs})\ \frac{\Psi;\Gamma, x{:}T \vdash M : S}{\Psi;\Gamma \vdash \lambda x{:}T.M : T \to S}$$

$$(\text{App})\ \frac{\Psi;\Gamma \vdash M : T \to S \qquad \Psi;\Gamma \vdash N : T}{\Psi;\Gamma \vdash MN : S}$$

Quote

$$(\text{Quo})\ \frac{\Psi; \Gamma \vdash M : T}{\Psi \vdash \text{`}\langle\Gamma\rangle M : [rg(\Gamma)]T}$$

- A binding form
- Term representation for hypothetical judgment
  - Γ : a list of assumptions
  - M : derivation tree
- Can be seen as “code with free variables”
  - Γ : a list of free variables
  - M : body of code

Unquote

$$(\mathrm{Unq})_l\ \frac{\Psi \vdash M : [T_1, \ldots, T_m]S \qquad \begin{array}{c}\Psi; \Gamma_l; \ldots; \Gamma_1 \vdash N_i : T_i \\ \text{for } 1 \leq i \leq m\end{array}}{\Psi; \Gamma_l; \ldots; \Gamma_1 \vdash\ ,_l\langle N_1, \ldots, N_m \rangle M : S}$$

$$\text{where} \begin{cases} l = 1 & \text{for K} & l = 0, 1 & \text{for T} \\ l \geq 1 & \text{for K4} & l \geq 0 & \text{for S4} \end{cases}$$

- An application form
- Instantiate the quoted hypothetical judgment
- Can be seen as "evaluation of the code through l-stages"
  - $N_1 \cdots N_n$ are the top-level definitions of the free variables
  - $l = 0 \rightarrow$ run-time code evaluation e.g. `eval` function

Substitution

$[N_1/x_1 \cdots N_n/x_n]_l$

$\Psi; \Gamma_l; \cdots; \Gamma_1 \vdash M{:}T$

- Substitute free variables at level l

(Substitution Lemma)

If $\Psi; x_1 : S_1, \ldots, x_m : S_m; \Gamma_{l-1}; \ldots; \Gamma_1 \vdash M : T$
and $\Psi; \Gamma \vdash N_i : S_i$ *for all* $1 \leq i \leq m$,
then $\Psi; \Gamma; \Gamma_{l-1}; \ldots; \Gamma_1 \vdash M[\sigma]_l : T$,
where $\sigma = N_1/x_1, \ldots, N_m/x_m$.

Level Substitution

$\uparrow_l^n$

- Merge the lth context with the $l+1$th context (when $n = 0$)
- Insert $n-1$ context upon the lth context (when $\mathrm{n} \geq 1$)

(Level Substitution Lemma)

(i) *For* $X \in \{T, S4\}$, *if* $\Psi; \Gamma_{l+1}; \Gamma_l; \ldots; \Gamma_1 \vdash_X M : T$,
then $\Psi; \Gamma_{l+1}, \Gamma_l; \ldots; \Gamma_1 \vdash_X M\uparrow_l^0 : T$.

(ii) *For* $X \in \{K4, S4\}$, *if* $\Psi; \Gamma_{l+1}; \Gamma_l; \ldots; \Gamma_1 \vdash_X M : T$,
then $\Psi; \Gamma_{l+1}; \Psi'; \Gamma_l; \ldots; \Gamma_1 \vdash_X M\uparrow_l^m : T$
where Ψ' *is size-*$(m-1)$ *stack of empty contexts.*

Reduction/Expansion Rules

- β-Reduction

$$(\lambda x : A.M)N \overset{R}{\Longrightarrow} M[N/x]_1$$

$$, k\langle N_1,\ldots,N_m\rangle\,‘\langle x_1{:}T_1,\ldots,x_m{:}T_m\rangle M \overset{R}{\Longrightarrow} M \uparrow_1^k [N_1/x_1,\ldots,N_m/x_m]_1$$

- η-Expansion

$$M \overset{E}{\Longrightarrow} \lambda x : T.Mx$$

$$\text{where } \Psi \vdash M : T \rightarrow S$$

$$M \overset{E}{\Longrightarrow} ‘\langle x_1{:}T_1,\ldots,x_m{:}T_m\rangle(\,,1\langle\vec{x}\rangle M)$$

$$\text{where } \Psi \vdash M : [T_1,\ldots,T_m]S$$

Example: or macro

```
(defmacro or (x y)
  `(if ,x true ,y))
```

```
or := λx:[A]bool. λy:[A]bool.
         `<w:A>(if ,1<w>(x)
                 then true
                 else ,1<w>(y))
```

Example: or macro

```
(or (= x x) false)
=> (if (= x x)
      true
      false)
```

```
,<w>(or `<w:A>(w=w) `<w:A>(false))
=> if w = w
     then true
     else false
```

Example: binding manipulation

```
(defmacro bind (y)
  `(lambda (x) ,y))
```

```
bind := λy:[A]bool.
          `<>(λx:A.,1<x>y)
```

Example: binding manipulation

```
(bind (= x x))
=> (lambda (x) (= x x))
```

```
,<>(bind (`<x:A>(x=x)))
=> λx:A. x=x
```

Agenda

- Background
- Logic
- Type System
- Future Plan/Related Work

Future Plan

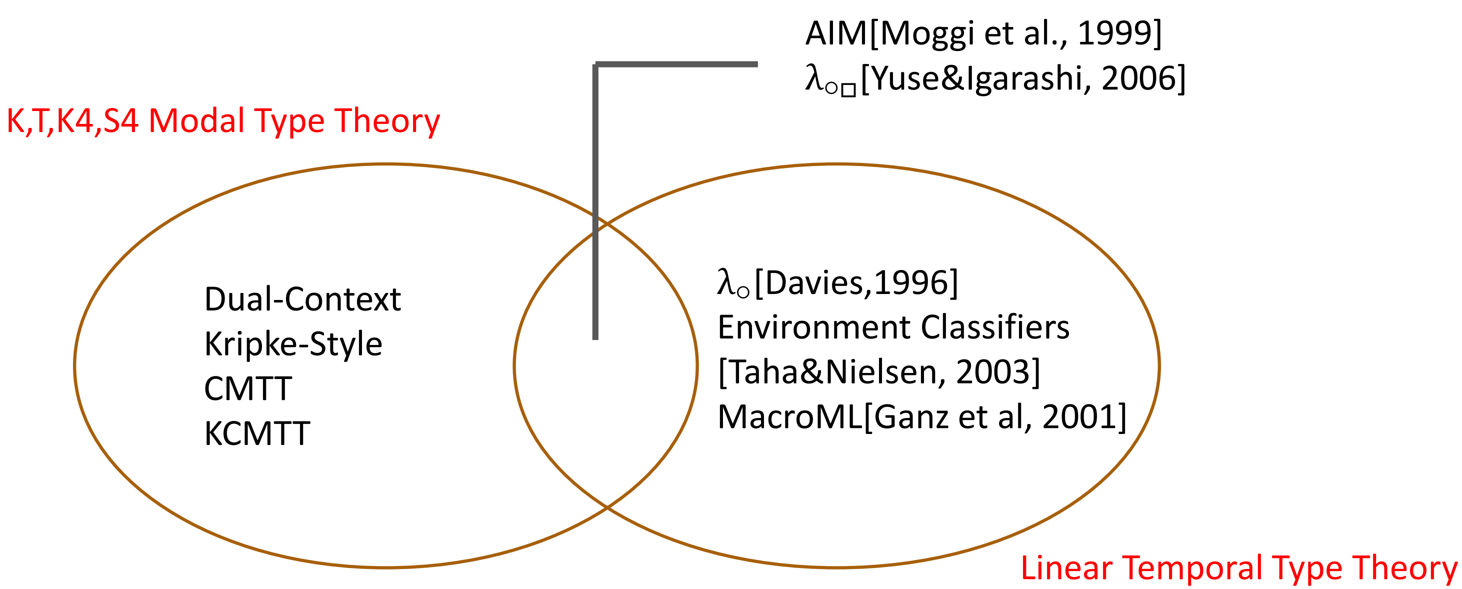
- Motivation: Reasoning syntactical metaprogramming
- Develop stronger type theory
  - Environment Polymorphism
- Develop a programming languages with type-safe syntactical metaprogramming
- Other problems
  - Confluency and Strong normalization
  - Categorical Semantics

Environment Polymorphism

- The type of or macro is too specific
  - $[]\text{bool} \rightarrow []\text{bool} \rightarrow []\text{bool}$
  - $[A]\text{bool} \rightarrow [A]\text{bool} \rightarrow [A]\text{bool}$
  - $[B, C]\text{bool} \rightarrow [B, C]\text{bool} \rightarrow [B, C]\text{bool}$
- Quantify the environment
  - $\forall \gamma.[\gamma]\text{bool} \rightarrow [\gamma]\text{bool} \rightarrow [\gamma]\text{bool}$
- Under construction

Related work: Linear Temporal Type Theory

- $\lambda_{\circ}$ [Davies, 1996] Correspond to linear temporal logic
  - Treats open code
  - Code generation is essentially hygienic
    - Cannot express the `bind` macro



Related work: λ_{open}^{sim}

- Proposed by Kim et al.(2006)
  - “A polymorphic modal type system for lisp-like multi-staged languages”
- Extend modal types to have context
  - $\square(\Gamma \triangleright A)$
  - Programming language with imperative features
- Undesirable nature as typed lambda calculi
  - α-renaming is restricted
  - Reduction rule is restricted

Summary

- We want type theory that reasons syntactical metaprogramming
- We proposed Kripke-style contextual modal type theory
  - Lisp-like quasi-quotation syntax
  - Contextual modal type
  - 4 variants(K, T, K4, S4)
  - Proved subject reduction of KCMTT
- Future work
  - Prove confluence and strong normalization
  - Develop the type theory
  - Develop the programming language