A Recursive Type System with Type Abbreviations

Hyeonseung Im

Keiko Nakata Sungwoo Park

PL Lab, POSTECH

ROSAEC Center 7th Workshop @ HKUST

January 19, 2012

Motivation: OCaml Recursive Modules

- OCaml recursive modules allow us to write mutually recursive definitions into separate modules.

```
module rec Expr : sig
  type t
end = struct
  type t = Var of string
         | LetExpr of Bind.t * t
         | ...
end
and Bind : sig
  type t
end = struct
  type t = LetBind of string * Expr.t
         | ...
end
```

Motivation: Non-contractive Type Cycles

- Contractive type cycles
  - *e.g.*, `type t = s * s`, `type s = t -> t`
- Non-contractive type cycles
  - *e.g.*, `type t = t`, `type s = u`, `type u = s`
- Why non-contractive type cycles matter? Because we cannot detect them accurately.

```
module rec M : sig type 'a t end =
  struct type 'a t = 'a N.t end
and N : sig type 'a t end =
  struct type 'a t = 'a M.t end
```

- To show soundness of OCaml Recursive Modules, we should be able to deal with non-contractive type cycles.

Type Language

- We prove soundness of a simple typed language with OCaml-style parameterized recursive type definitions, possibly non-contractive.

type names	s, t, u		
kinds	κ	$::=$	$1 \mid 1 \to 1$
types	τ, σ	$::=$	unit
		$\mid$	$\alpha \mid \beta \mid \gamma \mid \cdots$
		$\mid$	$\tau \to \sigma$
		$\mid$	$\tau_1 * \tau_2$
		$\mid$	t
		$\mid$	$\tau\ t$
type definitions	D	$::=$	type $t :: 1 = \tau$
		$\mid$	type $\alpha\ t :: 1 \to 1 = \tau$

- Challenge: To define a type equivalence relation being able to handle non-contractive type cycles.

Well-formed Types

- Given a set of mutually recursive type definitions, let us define well-formedness rules and a type equivalence relation.

Set of type definitions	Δ	$::= \quad \cdot \mid \Delta, D$
Type contexts	Σ	$::= \quad \cdot \mid \Sigma, \alpha \text{ type}$
Well-formed types	$\Delta; \Sigma \vdash \tau :: 1$	

$$\frac{}{\Delta; \Sigma \vdash \mathsf{unit} :: 1} \qquad \frac{\alpha \text{ type} \in \Sigma}{\Delta; \Sigma \vdash \alpha :: 1} \qquad \frac{\Delta; \Sigma \vdash \tau :: 1 \quad \Delta; \Sigma \vdash \sigma :: 1}{\Delta; \Sigma \vdash \tau \to \sigma :: 1}$$

$$\frac{\Delta; \Sigma \vdash \tau :: 1 \quad \Delta; \Sigma \vdash \sigma :: 1}{\Delta; \Sigma \vdash \tau * \sigma :: 1} \qquad \frac{\Delta \ni \mathsf{type}\ t :: 1 = \tau}{\Delta; \Sigma \vdash t :: 1}$$

$$\frac{\Delta \ni \mathsf{type}\ \alpha\ t :: 1 \to 1 = \sigma \quad \Delta; \Sigma \vdash \tau :: 1}{\Delta; \Sigma \vdash \tau\ t :: 1}$$

Well-formed Type Definitions

Well-formed definitions	$\Delta \vdash D \text{ wf}$
Well-formed sets	$\vdash \Delta \text{ wf}$

$$\frac{\Delta; \cdot \vdash \tau :: 1}{\Delta \vdash \text{type } t :: 1 = \tau \text{ wf}} \qquad \frac{\Delta; \alpha \text{ type} \vdash \tau :: 1}{\Delta \vdash \text{type } \alpha\ t :: 1 \to 1 = \tau \text{ wf}}$$

$$\frac{\text{BN}(\Delta) \text{ distinct} \quad \forall D \in \Delta,\ \Delta \vdash D \text{ wf}}{\Delta \vdash D \text{ wf}}$$

$$\begin{aligned}
\text{BN}(\Delta, \text{type } t :: 1 = \tau) &\triangleq \text{BN}(\Delta) \cup \{t\} \\
\text{BN}(\Delta, \text{type } \alpha\ t :: 1 \to 1 = \tau) &\triangleq \text{BN}(\Delta) \cup \{t\} \\
\text{BN}(\cdot) &\triangleq \emptyset
\end{aligned}$$

Type Equivalence

- To account for infinite unfolding of type abbreviations, we use coinductive type equivalence $\Delta; \Gamma \vdash \tau \equiv \sigma :: 1$.
  - e.g., `type s = t * u`, `type t = u * s`, `type u = s * t`
- A wrong attempt:

$$\frac{\Delta \ni \text{type } t = \tau \quad \Delta; \Sigma \vdash \tau \equiv \sigma :: 1}{\Delta; \Sigma \vdash t \equiv \sigma :: 1}$$

- This simple approach fails if $\tau = t$ and $\sigma = \text{unit}$.

$$\frac{\Delta \ni \text{type } t = t \quad \Delta; \Sigma \vdash t \equiv \text{unit} :: 1}{\Delta; \Sigma \vdash t \equiv \text{unit} :: 1}$$

- Apply coinduction if both types unfold.
- Otherwise apply induction.

Type Equivalence (Inductive Interpretation)

Inductive type equivalence $\Delta; \Gamma \vdash \tau = \sigma :: 1$

$$\frac{}{\Delta; \Sigma \vdash \mathsf{unit} = \mathsf{unit} :: 1} \qquad \frac{\alpha \in \Sigma}{\Delta; \Sigma \vdash \alpha = \alpha :: 1}$$

$$\frac{\Delta; \Sigma \vdash \tau_1 \equiv \sigma_1 :: 1 \quad \Delta; \Sigma \vdash \tau_2 \equiv \sigma_2 :: 1}{\Delta; \Sigma \vdash \tau_1 \to \tau_2 = \sigma_1 \to \sigma_2 :: 1}$$

$$\frac{\Delta; \Sigma \vdash \tau_1 \equiv \sigma_1 :: 1 \quad \Delta; \Sigma \vdash \tau_2 \equiv \sigma_2 :: 1}{\Delta; \Sigma \vdash \tau_1 * \tau_2 = \sigma_1 * \sigma_2 :: 1}$$

$$\frac{}{\Delta; \Sigma \vdash t = t :: 1} \qquad \frac{\Delta; \Sigma \vdash \tau \equiv \tau' :: 1}{\Delta; \Sigma \vdash \tau\ t = \tau'\ t :: 1}$$

Type Equivalence (Coinductive Interpretation)

- Unfolding type abbreviations:

$$\frac{\Delta \ni \text{type } t :: 1 = \sigma}{\Delta \vdash t \rightharpoonup \sigma} \qquad \frac{\Delta \ni \text{type } \alpha\ t :: 1 \to 1 = \sigma}{\Delta \vdash \tau\ t \rightharpoonup \{\tau/\alpha\}\sigma}$$

- $\Delta \vdash \tau \rightharpoonup^* \tau'$ is the reflexive and transitive closure of a single transition.

- Coinductive type equivalence:

$$\frac{\Delta \vdash \tau \rightharpoonup \tau' \quad \Delta \vdash \sigma \rightharpoonup \sigma' \quad \Delta; \Sigma \vdash \tau' \equiv \sigma' :: 1}{\Delta; \Sigma \vdash \tau \equiv \sigma :: 1}$$

$$\frac{\Delta \vdash \tau \rightharpoonup^* \tau' \quad \Delta \vdash \sigma \rightharpoonup^* \sigma' \quad \Delta; \Sigma \vdash \tau' = \sigma' :: 1}{\Delta; \Sigma \vdash \tau \equiv \sigma :: 1}$$

Conclusion and Future Work

- Studied a recursive type system with type abbreviations and proved its soundness.
- Am mechanizing the soundness proof using Coq.
  - Difficulty: Formalizing mixed induction-coinduction
  - Idea: Parameterizing an inductive definition by a coinductive definition
- Will extend the system with abstract types.

Thank you!

Questions?