A Recursive Type System with Type Abbreviations and Abstract Types

타입에 이름 붙이기와 타입의 속내용 감추기를 지원하는 재귀 타입 시스템

임현승, Keiko Nakata, 박성우

제 8회 ROSAEC Center Workshop @ 이천, 25-28 July 2012

Recursive Types

```
type 'a tree = Leaf of 'a
             | Node of 'a tree * 'a * 'a tree
```

- **Iso-recursive types**

  μα.T $\xrightarrow{\text{unfold}}$ {α → μα.T} T
  {α → μα.T} T $\xrightarrow{\text{fold}}$ μα.T

- **Equi-recursive types**

  μα.T ≡ {α → μα.T} T

- Structural polymorphism, e.g., polymorphic variants or objects, supported in OCaml requires **structural type equivalence** (equi-recursive types).

Non-contractive Types

- A type is **non-contractive** if unfolding type definitions **diverges** and is **not guarded** by a type constructor
  – `type t = t, type s = u and u = s`
- Contractive types
  – `type t = int, type 'a t = 'a, type t = t * t`
- We cannot detect non-contractive types accurately.

```
module rec P : sig type t end =
  struct type t = Q.t end
and Q : sig type t end =
  struct type t = P.t end
```

- **Our type equivalence relation should be able to handle non-contractive types.**

Abstract Types, Signature Sealing, Non-contractive Types

- Abstract types by signature sealing in ML

```
module M = struct          module type S = sig
  type 'a t = 'a             type 'a t
end                        end
module M = (M : S)         (* signature sealing *)
```

- Non-contractive types in a signature are a source of **type unsoundness**

```
module M = struct          module type S = sig
  type t = int               type t = t
  type s = bool              type s = s
  let succ x = x + 1         val succ : t -> t
  let bval = true            val bval : s
end                        end
module M = (M : S)
let x = M.succ M.sval  (* run-time error *)
```

- **Disallow non-contractive types in the sealed signature.**

Features of Our Recursive Type System

- Equi-recursive types, structural type equivalence
- **Type parameters, non-contractive types in the implementation, abstract types** (supported in OCaml, but no sound type theory)
- Type equivalence, contractiveness defined in **mixed induction and coinduction**

Syntax & Additional Judgments

type name	s,t,u		
type	τ,σ	::=	unit $\mid \alpha \mid \tau\to\sigma \mid \tau\, t$
term	e	::=	$() \mid a \mid x \mid \lambda a:\tau.e \mid e_1\, e_2 \mid$ fix $a:\tau.e \mid l$
specification	D	::=	type $\alpha\, t \mid$ type $\alpha\, t = \tau \mid$ val $l : \tau$
definition	d_τ	::=	type $\alpha\, t = \tau$
	d_e	::=	let $l = e$
signature	S	::=	$\cdot \mid S, D$
structure	M	::=	$(\overline{d_\tau}, \overline{d_e})$
program	P	::=	$(M,S,e) \mid (M,e)$
value context	Γ	::=	$\cdot \mid \Gamma, x:\tau$
type variable set	Σ	::=	$\cdot \mid \{\alpha\}$
well-formedness	$S;\alpha \vdash \tau$ type $\quad S \vdash D$ ok $\quad S$ ok		
membership	$S \ni$ type $\alpha\, t = \sigma$		
subtyping	$S_1 \le S_2 \quad S \vdash D_1 \le D_2$		
well-typedness	$\vdash P : (S,\tau) \quad \vdash M : S \quad S \vdash \overline{d_e} : S_e \quad S;\Gamma \vdash e : \tau$		
reduction	$P \longmapsto P' \quad M \longmapsto M' \quad \overline{d_v} \vdash e \longmapsto e'$		

Type Equivalence

Unfolding $\boxed{S \vdash \tau \rightharpoonup \sigma}$

$$\frac{S \ni \text{type } \alpha\, t = \sigma}{S \vdash \tau\, t \rightharpoonup \{\alpha \mapsto \tau\}\sigma}\ \text{unfold}$$

Coinductive type equivalence $\boxed{S;\Sigma \vdash \tau_1 \equiv \tau_2}$

$$\frac{R \subseteq \equiv \quad S;\Sigma \vdash \tau \stackrel{R}{=} \sigma}{S;\Sigma \vdash \tau \equiv \sigma}\ \text{eq-ind} \qquad \frac{S \vdash \tau \rightharpoonup \tau' \quad S \vdash \sigma \rightharpoonup \sigma' \quad S;\Sigma \vdash \tau' \equiv \sigma'}{S;\Sigma \vdash \tau \equiv \sigma}\ \text{eq-coind}$$

Inductive type equivalence $\boxed{S;\Sigma \vdash \tau_1 \stackrel{R}{=} \tau_2}$

$$\frac{}{S;\Sigma \vdash \text{unit} \stackrel{R}{=} \text{unit}}\ \text{eq-unit} \qquad \frac{\alpha \in \Sigma}{S;\Sigma \vdash \alpha \stackrel{R}{=} \alpha}\ \text{eq-var}$$

$$\frac{S;\Sigma \vdash \tau_1\, R\, \sigma_1 \quad S;\Sigma \vdash \tau_2\, R\, \sigma_2}{S;\Sigma \vdash \tau_1 \to \tau_2 \stackrel{R}{=} \sigma_1 \to \sigma_2}\ \text{eq-fun} \qquad \frac{S \ni \text{type } \alpha\, t \quad S;\Sigma \vdash \tau\, R\, \sigma}{S;\Sigma \vdash \tau\, t \stackrel{R}{=} \sigma\, t}\ \text{eq-abs}$$

$$\frac{\Delta \vdash \tau \rightharpoonup \tau' \quad \Delta;\Sigma \vdash \tau' \stackrel{R}{=} \sigma}{\Delta;\Sigma \vdash \tau \stackrel{R}{=} \sigma}\ \text{eq-lunfold} \qquad \frac{S \vdash \sigma \rightharpoonup \sigma' \quad S;\Sigma \vdash \tau \stackrel{R}{=} \sigma'}{S;\Sigma \vdash \tau \stackrel{R}{=} \sigma}\ \text{eq-runfold}$$

Contractive Types and Signatures

Contractive types $\boxed{S \Downarrow \tau}$ $\boxed{S \downarrow_C \tau}$

$$\frac{C \subseteq \Downarrow \quad S \downarrow_C \tau}{S \Downarrow \tau}\ \text{ctr-coind} \qquad \frac{}{S \downarrow_C \text{unit}}\ \text{ctr-unit} \qquad \frac{}{S \downarrow_C \alpha}\ \text{ctr-var}$$

$$\frac{(S,\tau) \in C \quad (S,\sigma) \in C}{S \downarrow_C \tau \to \sigma}\ \text{ctr-fun} \qquad \frac{S \ni \text{type } \alpha\, t \quad S \downarrow_C \tau}{S \downarrow_C \tau\, t}\ \text{ctr-abs} \qquad \frac{S \vdash \tau \rightharpoonup \sigma \quad S \downarrow_C \sigma}{S \downarrow_C \tau}\ \text{ctr-type}$$

Contractive signatures $\boxed{S \Downarrow}$

$$\frac{\text{BN}(S) \text{ distinct} \quad \forall (\text{type } \alpha\, t = \tau) \in S,\ S \Downarrow \tau}{S \Downarrow}\ \text{ctr-sig}$$

Type Soundness

value	v	::=	$() \mid \lambda a:\tau.e$
definition value	d_v	::=	let $l = v$
module value	V	::=	$(\overline{d_\tau}, \overline{d_v})$
program value	P_v	::=	(V, v)

Theorem A.1 (**Progress**)
If $\vdash P : (S,\tau)$, then either P is a program value or there exists P' such that $P \longmapsto P'$.

Proof
By induction on $\vdash P : (S,\tau)$. □

Theorem A.2 (**Preservation**)
1. If $\vdash (\overline{d_\tau}, \overline{d_v}) : S$, $S;\cdot \vdash e : \tau$, and $\overline{d_v} \vdash e \longmapsto e'$, then $S;\cdot \vdash e' : \tau$.
2. If $\vdash M : S$ and $M \longmapsto M'$, then $\vdash M' : S$.
3. If $P = (M,e)$, $\vdash P : (S,\tau)$ and $P \longmapsto P'$, then $\vdash P' : (S,\tau)$.
4. If $\vdash (M,S,e) : (S,\tau)$, then there exists S' such that $\vdash M : S'$, $S' \le S$, and $S';\cdot \vdash e : \tau$.

Proof
1. By induction on a derivation of $S;\cdot \vdash e : \tau$.
2. By case analysis using (1).
3. By case analysis using (1) and (2).
4. By using the signature elimination lemma.

□

Key Difficulty in Soundness Proofs

$$\frac{\vdash M : S' \quad S \Downarrow \quad S' \le S \quad S;\cdot \vdash e : \tau}{\vdash (M,S,e) : (S,\tau)}\ \text{typ-prog-seal} \qquad \frac{\vdash M : S \quad S;\cdot \vdash e : \tau}{\vdash (M,e) : (S,\tau)}\ \text{typ-prog}$$

Lemma A.3 (**Signature elimination**)
If $\vdash (M,S,e) : (S,\tau)$, then $\exists S'$ such that $\vdash (M,e) : (S',\tau)$ and $S' \le S$.

Lemma A.4 (**Typing is preserved by signature elimination**)
If $S_1 \le S_2$, $S_2 \Downarrow$ and $S_2;\Gamma \vdash e : \tau$, then $S_1;\Gamma \vdash e : \tau$.

Lemma A.5 (**Type equivalence is preserved by signature elimination**)
If $S_1 \le S_2$, $S_2 \Downarrow$ and $S_2;\Sigma \vdash \tau \equiv \sigma$, then $S_1;\Sigma \vdash \tau \equiv \sigma$.

Lemma A.6 (**Well-formed types are contractive**)
Suppose S ok, $S \Downarrow$, and $S;\Sigma \vdash \tau$ type. Then $S \Downarrow \tau$.

Proof
The proof is by coinduction. The derivation tree below illustrates the key idea of the proof where relation C is defined as $\{(S_0,\tau_0) \mid S_0 \text{ ok},\ S_0 \Downarrow, \text{ and } S_0;\Sigma_0 \vdash \tau_0 \text{ type}\}$.

$$\dfrac{(1)\ C \subseteq \Downarrow \qquad \dfrac{S \vdash \tau \rightharpoonup \sigma \qquad \dfrac{\vdots}{S \downarrow_C \sigma}}{S \downarrow_C \tau}}{S \Downarrow \tau}$$

where

$$\dfrac{S \ni \text{type } \alpha_i\, t_i \qquad \dfrac{\vdots}{S \downarrow_C \sigma_i}}{S \downarrow_C \sigma_i\, t_i} \qquad\qquad (2)\ \dfrac{S \ni \text{type } \alpha_j\, t_j \quad S \downarrow_C \sigma_j \quad \sigma_j = \text{unit},\ \beta,\ \tau_1 \to \tau_2, \text{ or } \tau_1 * \tau_2}{S \downarrow_C \sigma_j\, t_j}$$

□

Contributions

- First sound type system with type parameters, non-contractive types, and abstract types
- Interesting proof techniques
- Whole system and proofs are formalized in Coq

Strong Contractiveness

Strong unfolding $\boxed{S \vdash \tau \Rightarrow \sigma}$

$$\frac{S \vdash \tau \Rightarrow \sigma}{S \vdash \tau\, t \Rightarrow \sigma\, t}\ \text{sunfold-abs} \qquad \frac{S \ni \text{type } \alpha\, t = \sigma}{S \vdash \tau\, t \Rightarrow \{\alpha \mapsto \tau\}\sigma}\ \text{sunfold-type}$$

Strong contractive types $\boxed{S \Downarrow^{\mathsf{s}} \tau}$ $\boxed{S \downarrow^{\mathsf{s}}_C \tau}$

$$\frac{C \subseteq \Downarrow^{\mathsf{s}} \quad S \vdash \tau \Rightarrow^* \sigma \quad S \downarrow^{\mathsf{s}}_C \sigma}{S \Downarrow^{\mathsf{s}} \tau}\ \text{sctr-coind} \qquad \frac{}{S \downarrow^{\mathsf{s}}_C \text{unit}}\ \text{sctr-unit} \qquad \frac{}{S \downarrow^{\mathsf{s}}_C \alpha}\ \text{sctr-var}$$

$$\frac{(S,\tau) \in C \quad (S,\sigma) \in C}{S \downarrow^{\mathsf{s}}_C \tau \to \sigma}\ \text{sctr-fun} \qquad \frac{S \ni \text{type } \alpha\, t \quad S \downarrow^{\mathsf{s}}_C \tau}{S \downarrow^{\mathsf{s}}_C \tau\, t}\ \text{sctr-abs}$$

Strong contractive signature $\boxed{S \Downarrow^{\mathsf{s}}}$

$$\frac{\text{BN}(S) \text{ distinct} \quad \forall (\text{type } \alpha\, t = \tau) \in S,\ S \Downarrow^{\mathsf{s}} \tau}{S \Downarrow^{\mathsf{s}}}\ \text{sctr-sig}$$

Lemma A.7 (**Equivalence between contractiveness and strong contractiveness**)
Suppose S ok. Then $S \Downarrow \tau$ if and only if $S \Downarrow^{\mathsf{s}} \tau$.

Proof
The proof is by induction nested into coinduction. □

Corollary A.8
$S \Downarrow$ if and only if $S \Downarrow^{\mathsf{s}}$.

Proof
Corollary of Lemma A.7 □

Type Soundness Again!

Original subtyping

$$\frac{S_1 \text{ ok} \quad S_2 \text{ ok} \quad \forall n \in \text{dom}(S_2),\ S_1 \vdash S_1(n) \le S_2(n)}{S_1 \le S_2}\ \text{sub-sig}$$

Refined subtyping

$$\frac{S_1 \text{ ok} \quad S_2 \text{ ok} \quad |\text{dom}(S_1)| = |\text{dom}(S_2)| \quad \forall n \in \text{dom}(S_2),\ S_1 \vdash S_1(n) \le S_2(n)}{S_1 \leqq S_2}\ \text{sub-sigeq}$$

Signature extension

$$\begin{array}{rcl} \mathit{SigExt}(S_1,S_2) & := & (S_1/S_2)^\circ \cup S_2 \\ S_1/S_2 & := & \{D_1 \mid D_1 \in S_1,\ \forall D_2 \in S_2,\ \text{BN}(D_1) \neq \text{BN}(D_2)\} \\ (D_1,\ldots,D_n)^\circ & := & (D_1)^\circ,\ldots,(D_n)^\circ \\ (\text{type } \alpha\, t = \tau)^\circ & := & \text{type } \alpha\, t \\ (D)^\circ & := & D \quad \text{where } D \text{ is not a type equation} \end{array}$$

Lemma A.9
If $S_1 \le S_2$ then $S_1 \leqq \mathit{SigExt}(S_1,S_2)$.

Proof
By the definition of *SigExt* and subtyping. □

Lemma A.10 (**Type equivalence is preserved by signature elimination**)
If $S_1 \leqq S_2$, $S_2 \Downarrow^{\mathsf{s}}$ and $S_2;\Sigma \vdash \tau \equiv \sigma$, then $S_1;\Sigma \vdash \tau \equiv \sigma$.

Proof
The proof is by coinduction and various properties on $S_2 \Downarrow^{\mathsf{s}}$ □

Lemma A.11 (**Typing is preserved by signature elimination**)
If $S_1 \leqq S_2$, $S_2 \Downarrow^{\mathsf{s}}$ and $S_2;\Gamma \vdash e : \tau$, then $S_1;\Gamma \vdash e : \tau$.

Proof
By rule induction on $S_2;\Gamma \vdash e : \tau$ □

Proof of Lemma A.3 (Signature elimination)

- $\vdash M : S'$, $S' \le S$, $S \Downarrow$, and $S;\cdot \vdash e : \tau$ — By inversion on $\vdash (M,S,e) : (S,\tau)$
- $S' \leqq \mathit{SigExt}(S',S)$ — By Lemma A.9 with $S' \le S$
- $\mathit{SigExt}(S',S) \Downarrow^{\mathsf{s}}$ — From $S \Downarrow$
- $\mathit{SigExt}(S',S);\cdot \vdash e : \tau$ — By weakening with $S;\cdot \vdash e : \tau$
- $S';\cdot \vdash e : \tau$ — By Lemma A.11
- $\vdash (M,e) : (S',\tau)$ — By the rule typ-prog with $\vdash M : S'$ and $S';\cdot \vdash e : \tau$

□