Type and Scope Preserving Semantics

Guillaume Allais

James Chapman   Conor McBride

University of Strathclyde

SPLS, 2015-10-21

Motivations

- Formally studying PLs
  - Representation of Terms / Typing derivations
  - With good properties: closed under renaming and substitution, normalising
  - Themselves with good properties
- Writing DSLs
  - Strong guarantees (type, scope safety)
  - With ASTs we can inspect (optimise, compile)

Simple Types

Minimal system: A record type, a sum type and function spaces.

```
data ty : Set where
  `Unit  : ty
  `Bool  : ty
  _`→_   : (σ τ : ty) → ty
```

```
data Con : Set where
  ε    : Con
  _•_  : Con → ty → Con
```

Deep Embedding - Variables

Typed de Bruijn indices

```
data _∈_ (σ : ty) : Con → Set where
  zero  : σ ∈ (Γ • σ)
  1+_   : σ ∈ Γ → σ ∈ (Γ • τ)
```

Deep Embedding - Terms

ASTs type and scope correct by construction

```
data _⊢_ (Γ : Con) : (σ : ty) → Set where
  `var    : (v : σ ∈ Γ) → Γ ⊢ σ
  _`$_    : (t : Γ ⊢ (σ `→ τ)) (u : Γ ⊢ σ) → Γ ⊢ τ
  `λ      : (t : Γ • σ ⊢ τ) → Γ ⊢ (σ `→ τ)
  `⟨⟩     : Γ ⊢ `Unit
  `tt `ff : Γ ⊢ `Bool
  `ifte   : (b : Γ ⊢ `Bool) (l r : Γ ⊢ σ) → Γ ⊢ σ
```

Goguen & McKinna: Conspicuously similar functions

$$
\begin{array}{ll}
\mathsf{ren} : (t : \Gamma \vdash \sigma)\ (\rho : \Delta\ [\ \mathsf{ren}\mathscr{E}\ ]\ \Gamma) \to \Delta \vdash \sigma & \\
\mathsf{ren}\ (\text{`var}\ v) & \rho = \mathsf{ren}[\![\mathsf{var}]\!]\ (\rho\ \_\ v) \\
\mathsf{ren}\ (t\ \text{`\$}\ u) & \rho = \mathsf{ren}\ t\ \rho\ \text{`\$}\ \mathsf{ren}\ u\ \rho \\
\mathsf{ren}\ (\text{`}\lambda\ t) & \rho = \text{`}\lambda\ (\mathsf{ren}\ t\ (\mathsf{renextend}\ \rho)) \\
\mathsf{ren}\ \text{`}\langle\rangle & \rho = \text{`}\langle\rangle \\
\mathsf{ren}\ \text{`tt} & \rho = \text{`tt} \\
\mathsf{ren}\ \text{`ff} & \rho = \text{`ff} \\
\mathsf{ren}\ (\text{`ifte}\ b\ l\ r) & \rho = \text{`ifte}\ (\mathsf{ren}\ b\ \rho)\ (\mathsf{ren}\ l\ \rho)\ (\mathsf{ren}\ r\ \rho)
\end{array}
$$

Goguen & McKinna: Conspicuously similar functions

$$
\begin{array}{ll}
\mathsf{sub} : (t : \Gamma \vdash \sigma)\ (\rho : \Delta\ [\ \mathsf{sub}\mathscr{E}\ ]\ \Gamma) \to \Delta \vdash \sigma & \\
\mathsf{sub}\ (\text{`var}\ v) & \rho = \mathsf{sub}[\![\mathsf{var}]\!]\ (\rho\ \_\ v) \\
\mathsf{sub}\ (t\ \text{`\$}\ u) & \rho = \mathsf{sub}\ t\ \rho\ \text{`\$}\ \mathsf{sub}\ u\ \rho \\
\mathsf{sub}\ (\text{`}\lambda\ t) & \rho = \text{`}\lambda\ (\mathsf{sub}\ t\ (\mathsf{subextend}\ \rho)) \\
\mathsf{sub}\ \text{`}\langle\rangle & \rho = \text{`}\langle\rangle \\
\mathsf{sub}\ \text{`tt} & \rho = \text{`tt} \\
\mathsf{sub}\ \text{`ff} & \rho = \text{`ff} \\
\mathsf{sub}\ (\text{`ifte}\ b\ l\ r) & \rho = \text{`ifte}\ (\mathsf{sub}\ b\ \rho)\ (\mathsf{sub}\ l\ \rho)\ (\mathsf{sub}\ r\ \rho)
\end{array}
$$

Factoring Out the Common Parts

```
record Syntactic (ℰ : (Γ : Con) (σ : ty) → Set) : Set where
  field embed  : (σ : ty) → σ ∈ Γ → ℰ Γ σ
        wk     : Γ ⊆ Δ → ℰ Γ σ → ℰ Δ σ
        ⟦var⟧  : ℰ Γ σ → Γ ⊢ σ
```

Implementing the traversal Once and For All

```
syn : (𝒮 : Syntactic ℰ) (t : Γ ⊢ σ) (ρ : Δ [ ℰ ] Γ) → Δ ⊢ σ
syn 𝒮 (`var v)       ρ = Syntactic.⟦var⟧ 𝒮 (ρ _ v)
syn 𝒮 (t `$ u)       ρ = syn 𝒮 t ρ `$ syn 𝒮 u ρ
syn 𝒮 (`λ t)         ρ = `λ (syn 𝒮 t (synextend 𝒮 ρ))
syn 𝒮 `⟨⟩            ρ = `⟨⟩
syn 𝒮 `tt            ρ = `tt
syn 𝒮 `ff            ρ = `ff
syn 𝒮 (`ifte b l r)  ρ = `ifte (syn 𝒮 b ρ) (syn 𝒮 l ρ) (syn 𝒮 r ρ)
```

```
synextend : (𝒮 : Syntactic ℰ) (ρ : Δ [ ℰ ] Γ) → Δ • σ [ ℰ ] Γ • σ
synextend 𝒮 ρ = [ ℰ ] ρ′ `• var
  where var = Syntactic.embed 𝒮 _ zero
        ρ′  = λ σ → Syntactic.wk 𝒮 (step refl) ∘ ρ σ
```

Is that it? Not quite.

- Other interesting instance?
- Properties of these traversals? (4 fusion lemmas)
- What if I don't program in Agda?
- Generic boilerplate for all syntaxes with binding?

Normalisation by Evaluation's "eval"

$$
\begin{array}{ll}
\mathsf{sem} : (t : \Gamma \vdash \sigma)\ (\rho : \Delta\ [\ \_\vDash^{\beta\iota\xi\eta}\_\ ]\ \Gamma) \to \Delta \vDash^{\beta\iota\xi\eta} \sigma & \\
\mathsf{sem}\ (\text{`}\mathsf{var}\ v) & \rho = \mathsf{sem}[\![\mathsf{var}]\!]\ (\rho\ \_\ v) \\
\mathsf{sem}\ (t\ \text{`}\$\ u) & \rho = \mathsf{sem}\ t\ \rho\ \$^{\beta\iota\xi\eta}\ \mathsf{sem}\ u\ \rho \\
\mathsf{sem}\ (\text{`}\lambda\ t) & \rho = \mathsf{sem}\lambda\ (\mathsf{sem}\ t)\ (\mathsf{semextend}\ \rho) \\
\mathsf{sem}\ \text{`}\langle\rangle & \rho = \langle\rangle \\
\mathsf{sem}\ \text{`}\mathsf{tt} & \rho = \text{`}\mathsf{tt} \\
\mathsf{sem}\ \text{`}\mathsf{ff} & \rho = \text{`}\mathsf{ff} \\
\mathsf{sem}\ (\text{`}\mathsf{ifte}\ b\ l\ r) & \rho = \mathsf{ifte}^{\beta\iota\xi\eta}\ (\mathsf{sem}\ b\ \rho)\ (\mathsf{sem}\ l\ \rho)\ (\mathsf{sem}\ r\ \rho)
\end{array}
$$

Normalisation by Evaluation's "eval"

$$
\begin{array}{ll}
\mathsf{sub} : (t : \Gamma \vdash \sigma)\ (\rho : \Delta\ [\ \mathsf{sub}\mathscr{E}\ ]\ \Gamma) \to \Delta \vdash \sigma & \\
\mathsf{sub}\ (\text{`var}\ v) & \rho = \mathsf{sub}[\![\mathsf{var}]\!]\ (\rho\ \_\ v) \\
\mathsf{sub}\ (t\ \text{`\$}\ u) & \rho = \mathsf{sub}\ t\ \rho\ \text{`\$}\ \mathsf{sub}\ u\ \rho \\
\mathsf{sub}\ (\text{`}\lambda\ t) & \rho = \text{`}\lambda\ (\mathsf{sub}\ t\ (\mathsf{subextend}\ \rho)) \\
\mathsf{sub}\ \text{`}\langle\rangle & \rho = \text{`}\langle\rangle \\
\mathsf{sub}\ \text{`tt} & \rho = \text{`tt} \\
\mathsf{sub}\ \text{`ff} & \rho = \text{`ff} \\
\mathsf{sub}\ (\text{`ifte}\ b\ l\ r) & \rho = \text{`ifte}\ (\mathsf{sub}\ b\ \rho)\ (\mathsf{sub}\ l\ \rho)\ (\mathsf{sub}\ r\ \rho)
\end{array}
$$

An Abstract Notion of Semantics

```
record Semantics (𝓔 𝓜 : Con → ty → Set) : Set where
  field
```

An Abstract Notion of Semantics

```
record Semantics (𝓔 𝓜 : Con → ty → Set) : Set where
  field

  wk     : Γ ⊆ Δ → 𝓔 Γ σ → 𝓔 Δ σ
  embed  : ∀ σ → σ ∈ Γ → 𝓔 Γ σ
  ⟦var⟧  : 𝓔 Γ σ → 𝓜 Γ σ
```

An Abstract Notion of Semantics

```
record Semantics (ℰ ℳ : Con → ty → Set) : Set where
  field

  wk     : Γ ⊆ Δ → ℰ Γ σ → ℰ Δ σ
  embed  : ∀ σ → σ ∈ Γ → ℰ Γ σ
  ⟦var⟧  : ℰ Γ σ → ℳ Γ σ

  ⟦λ⟧  : (t : ∀ Δ → Γ ⊆ Δ → ℰ Δ σ → ℳ Δ τ) → ℳ Γ (σ `→ τ)
```

An Abstract Notion of Semantics

```
record Semantics (𝓔 𝓜 : Con → ty → Set) : Set where
  field

  wk     : Γ ⊆ Δ → 𝓔 Γ σ → 𝓔 Δ σ
  embed  : ∀ σ → σ ∈ Γ → 𝓔 Γ σ
  ⟦var⟧  : 𝓔 Γ σ → 𝓜 Γ σ

  ⟦λ⟧  : (t : ∀ Δ → Γ ⊆ Δ → 𝓔 Δ σ → 𝓜 Δ τ) → 𝓜 Γ (σ `→ τ)

  _⟦$⟧_  : 𝓜 Γ (σ `→ τ) → 𝓜 Γ σ → 𝓜 Γ τ
```

An Abstract Notion of Semantics

```
record Semantics (𝓔 𝓜 : Con → ty → Set) : Set where
  field

  wk      : Γ ⊆ Δ → 𝓔 Γ σ → 𝓔 Δ σ
  embed   : ∀ σ → σ ∈ Γ → 𝓔 Γ σ
  ⟦var⟧   : 𝓔 Γ σ → 𝓜 Γ σ

  ⟦λ⟧   : (t : ∀ Δ → Γ ⊆ Δ → 𝓔 Δ σ → 𝓜 Δ τ) → 𝓜 Γ (σ `→ τ)

  _⟦$⟧_  : 𝓜 Γ (σ `→ τ) → 𝓜 Γ σ → 𝓜 Γ τ

  ⟦⟨⟩⟧    : 𝓜 Γ `Unit
  ⟦tt⟧    : 𝓜 Γ `Bool
  ⟦ff⟧    : 𝓜 Γ `Bool
  ⟦ifte⟧  : (b : 𝓜 Γ `Bool) (l r : 𝓜 Γ σ) → 𝓜 Γ σ
```

And a Fundamental Lemma

```
lemma : (t : Γ ⊢ σ) (ρ : Δ [ 𝓔 ] Γ) → 𝓜 Δ σ
lemma (`var v)       ρ = ⟦var⟧ $ ρ _ v
lemma (t `$ u)       ρ = lemma t ρ ⟦$⟧ lemma u ρ
lemma (`λ t)         ρ = ⟦λ⟧ λ inc u → lemma t $ [ 𝓔 ] wk[ wk ] inc ρ `∙ u
lemma `⟨⟩            ρ = ⟦⟨⟩⟧
lemma `tt            ρ = ⟦tt⟧
lemma `ff            ρ = ⟦ff⟧
lemma (`ifte b l r)  ρ = ⟦ifte⟧ (lemma b ρ) (lemma l ρ) (lemma r ρ)
```

Various Instances

Renaming : Semantics (flip _∈_) _⊢_
Substitution : Semantics _⊢_ _⊢_
Printing : Semantics Name Printer
Normalise$^{\beta\iota\xi\eta}$: Semantics _⊨$^{\beta\iota\xi\eta}$_ _⊨$^{\beta\iota\xi\eta}$_

Is that it? Not quite.

- Other interesting instance?
- Properties of these traversals? (4 fusion lemmas)
- What if I don't program in Agda?
- Generic boilerplate for all syntaxes with binding?

A Relational Interpretation

```
record Synchronisable
  (𝒮ᴬ : Semantics ℰᴬ ℳᴬ) (𝒮ᴮ : Semantics ℰᴮ ℳᴮ)
  (ℰᴿ  : ℰᴬ Γ σ → ℰᴮ Γ σ → Set)
  (ℳᴿ  : ℳᴬ Γ σ → ℳᴮ Γ σ → Set) : Set where
```

A Relational Interpretation

```
record Synchronisable
  (𝒮ᴬ : Semantics ℰᴬ ℳᴬ) (𝒮ᴮ : Semantics ℰᴮ ℳᴮ)
  (ℰᴿ  : ℰᴬ Γ σ → ℰᴮ Γ σ → Set)
  (ℳᴿ  : ℳᴬ Γ σ → ℳᴮ Γ σ → Set) : Set where

  ℰᴿ_wk  :  (inc : Δ ⊆ Θ) (ρᴿ : '∀[ ℰᴬ , ℰᴮ ] ℰᴿ ρᴬ ρᴮ) →
            '∀[ ℰᴬ , ℰᴮ ] ℰᴿ (wk[ 𝒮ᴬ.wk ] inc ρᴬ) (wk[ 𝒮ᴮ.wk ] inc ρᴮ)
```

A Relational Interpretation

```
record Synchronisable
  (𝒮ᴬ : Semantics ℰᴬ ℳᴬ) (𝒮ᴮ : Semantics ℰᴮ ℳᴮ)
  (ℰᴿ  : ℰᴬ Γ σ → ℰᴮ Γ σ → Set)
  (ℳᴿ  : ℳᴬ Γ σ → ℳᴮ Γ σ → Set) : Set where

  ℰᴿ_wk  :  (inc : Δ ⊆ Θ) (ρᴿ : `∀[ ℰᴬ , ℰᴮ ] ℰᴿ ρᴬ ρᴮ) →
            `∀[ ℰᴬ , ℰᴮ ] ℰᴿ (wk[ 𝒮ᴬ.wk ] inc ρᴬ) (wk[ 𝒮ᴮ.wk ] inc ρᴮ)

  R⟦var⟧  :  (v : σ ∈ Γ) (ρᴿ : `∀[ ℰᴬ , ℰᴮ ] ℰᴿ ρᴬ ρᴮ) →
             ℳᴿ (𝒮ᴬ.⟦var⟧ (ρᴬ σ v)) (𝒮ᴮ.⟦var⟧ (ρᴮ σ v))
```

A Relational Interpretation

```
record Synchronisable
  (𝒮ᴬ : Semantics ℰᴬ ℳᴬ) (𝒮ᴮ : Semantics ℰᴮ ℳᴮ)
  (ℰᴿ  : ℰᴬ Γ σ → ℰᴮ Γ σ → Set)
  (ℳᴿ  : ℳᴬ Γ σ → ℳᴮ Γ σ → Set) : Set where
  ℰᴿ_wk  :  (inc : Δ ⊆ Θ) (ρᴿ : `∀[ ℰᴬ , ℰᴮ ] ℰᴿ ρᴬ ρᴮ) →
            `∀[ ℰᴬ , ℰᴮ ] ℰᴿ (wk[ 𝒮ᴬ.wk ] inc ρᴬ) (wk[ 𝒮ᴮ.wk ] inc ρᴮ)
  R⟦var⟧  :  (v : σ ∈ Γ) (ρᴿ : `∀[ ℰᴬ , ℰᴮ ] ℰᴿ ρᴬ ρᴮ) →
             ℳᴿ (𝒮ᴬ.⟦var⟧ (ρᴬ σ v)) (𝒮ᴮ.⟦var⟧ (ρᴮ σ v))
  R⟦λ⟧  :  (fᴿ : (pr : Γ ⊆ Δ) (uᴿ : ℰᴿ uᴬ uᴮ) → ℳᴿ (fᴬ pr uᴬ) (fᴮ pr uᴮ))
           → ℳᴿ (𝒮ᴬ.⟦λ⟧ fᴬ) (𝒮ᴮ.⟦λ⟧ fᴮ)
```

And a Fundamental Lemma

$$
\begin{array}{ll}
\mathsf{lemma} : & (t : \Gamma \vdash \sigma)\ (\rho^{\mathsf{R}} : \text{`}\forall[\ \mathscr{E}^{\mathsf{A}}\ ,\ \mathscr{E}^{\mathsf{B}}\ ]\ \mathscr{E}^{\mathsf{R}}\ \rho^{\mathsf{A}}\ \rho^{\mathsf{B}}) \to \\
& \mathscr{M}^{\mathsf{R}}\ (\mathcal{S}^{\mathsf{A}} \vDash [\![\ t\ ]\!]\ \rho^{\mathsf{A}})\ (\mathcal{S}^{\mathsf{B}} \vDash [\![\ t\ ]\!]\ \rho^{\mathsf{B}})
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{lemma}\ (\text{`}\mathsf{var}\ v) & \rho^{\mathsf{R}} = \mathsf{R}[\![\mathsf{var}]\!]\ v\ \rho^{\mathsf{R}} \\
\mathsf{lemma}\ (f\ \text{`}\$\ t) & \rho^{\mathsf{R}} = \mathsf{R}[\![\$]\!]\ (\mathsf{lemma}\ f\ \rho^{\mathsf{R}})\ (\mathsf{lemma}\ t\ \rho^{\mathsf{R}}) \\
\mathsf{lemma}\ (\text{`}\lambda\ t) & \rho^{\mathsf{R}} = \mathsf{R}[\![\lambda]\!]\ \lambda\ inc\ u^{\mathsf{R}} \to \mathsf{lemma}\ t\ ([\ \mathscr{E}^{\mathsf{A}}\ ,\ \mathscr{E}^{\mathsf{B}}\ ,\ \mathscr{E}^{\mathsf{R}}\ ]\ \mathscr{E}^{\mathsf{R}}_{\mathsf{wk}}\ inc\ \rho \\
\mathsf{lemma}\ \text{`}\langle\rangle & \rho^{\mathsf{R}} = \mathsf{R}[\![\langle\rangle]\!] \\
\mathsf{lemma}\ \text{`}\mathsf{tt} & \rho^{\mathsf{R}} = \mathsf{R}[\![\mathsf{tt}]\!] \\
\mathsf{lemma}\ \text{`}\mathsf{ff} & \rho^{\mathsf{R}} = \mathsf{R}[\![\mathsf{ff}]\!] \\
\mathsf{lemma}\ (\text{`}\mathsf{ifte}\ b\ l\ r) & \rho^{\mathsf{R}} = \mathsf{R}[\![\mathsf{ifte}]\!]\ (\mathsf{lemma}\ b\ \rho^{\mathsf{R}})\ (\mathsf{lemma}\ l\ \rho^{\mathsf{R}})\ (\mathsf{lemma}\ r\ \rho^{\mathsf{R}})
\end{array}
$$

An interesting corollary

$$\mathsf{SynchronisableNormalise} : \mathsf{Synchronisable}\ \mathsf{Normalise}^{\beta\iota\xi\eta}\ \mathsf{Normalise}^{\beta\iota\xi\eta}$$
$$(\mathsf{EQREL}\ \_\ \_)\ (\mathsf{EQREL}\ \_\ \_)$$

$$\mathsf{refl}^{\beta\iota\xi\eta} : (t : \Gamma \vdash \sigma)\ (\rho^{\mathsf{R}} : \text{`}\forall[\ \_\models^{\beta\iota\xi\eta}\_\ ,\ \_\ ]\ (\mathsf{EQREL}\ \_\ \_)\ \rho^{\mathsf{A}}\ \rho^{\mathsf{B}}) \to$$
$$\mathsf{EQREL}\ \Delta\ \sigma\ (\mathsf{Normalise}^{\beta\iota\xi\eta} \models [\![\ t\ ]\!]\ \rho^{\mathsf{A}})\ (\mathsf{Normalise}^{\beta\iota\xi\eta} \models [\![\ t\ ]\!]\ \rho^{\mathsf{B}})$$
$$\mathsf{refl}^{\beta\iota\xi\eta}\ t\ \rho^{\mathsf{R}} = \mathsf{lemma}\ t\ \rho^{\mathsf{R}}\ \mathsf{where\ open\ Synchronised\ SynchronisableNormalise}$$

Is that it? Not quite.

- Other interesting instance?
- Properties of these traversals? (4 fusion lemmas)
- What if I don't program in Agda?
- Generic boilerplate for all syntaxes with binding?

Using this somewhere else

The programming part of this talk can be implemented in Haskell:
`https://github.com/gallais/type-scope-semantics/`

Is that it? Not quite.

- Other interesting instance?
- Properties of these traversals? (4 fusion lemmas)
- What if I don't program in Agda?
- Generic boilerplate for all syntaxes with binding?