Syntactic Subsets for Decidable Bounded Polymorphism

Julian Mackay

Animals

Animals

type Animal = {def eat(f : Food)}

Animals

```
type Animal = {def eat(f : Food)}

def feed(a : Animal, f : Food) = a.eat(f)
```

Animals

```
type Animal = {def eat(f : Food)}

def feed(a : Animal, f : Food) = a.eat(f)

type Herbivore = {def eat(f : Plant)}
```

Animals

```
type Animal = {def eat(f : Food)}

def feed(a : Animal, f : Food) = a.eat(f)

type Herbivore = {def eat(f : Plant)}

type Carnivore= {def eat(f : Animal)}
```

Animals

type Animal = {def eat(f : Food)}

def feed(a : Animal, f : Food) = a.eat(f)

type Herbivore = {def eat(f : Plant)}

type Carnivore= {def eat(f : Animal)}

doesn’t work!!!! ----------> feed(antelope, lion)

Polymorphism

type Animal[F] = {def eat (f : F)}

Polymorphism

type Animal[F] = {def eat (f : F)}

type Herbivore = Animal[Plant]

Polymorphism

```
type Animal[F] = {def eat (f : F)}

type Herbivore = Animal[Plant]

type Carnivore = Animal[Animal[_]]
```

Polymorphism

```
type Animal[F] = {def eat (f : F)}

type Herbivore = Animal[Plant]

type Carnivore = Animal[Animal[_]]

def feed[F](a : Animal[F], f : F) = a.eat(f)
```

Polymorphism

```
type Animal[F] = {def eat (f : F)}

type Herbivore = Animal[Plant]

type Carnivore = Animal[Animal[_]]

def feed[F](a : Animal[F], f : F) = a.eat(f)
```

still not quite complete, there’s nothing constraining F

Polymorphism

type Animal[F] = {def eat (f : F)}

type Herbivore = Animal[Plant]

type Carnivore = Animal[Animal[_]]

def feed[F](a : Animal[F], f : F) = a.eat(f)

still not quite complete, there's nothing constraining F

----------> Animal[Emptiness] doesn't make sense

Bounded Polymorphism

```
type Animal[F <: Food] = {def eat (f : F)}
```

System $F_{<:}$

τ	::=	**System $F_{<:}$ Type**
	$\top$	*top*
	α	*variable*
	$\tau \rightarrow \tau$	*arrow*
	$\forall(\alpha \leqslant \tau).\tau$	*all*

Figure 1. System $F_{<:}$ Type Syntax

System $F_{<:}$

τ	::=	**System $F_{<:}$ Type**
	$\top$	*top*
	α	*variable*
	$\tau \rightarrow \tau$	*arrow*
	$\forall(\alpha \leqslant \tau).\tau$	*all*

Figure 1. System $F_{<:}$ Type Syntax

α can be used in τ

System $F_{<:}$

$$\Gamma \vdash \tau \ <: \ \top \quad (S^N\text{-Top}) \qquad \Gamma \vdash \alpha \ <: \ \alpha \quad (S^N\text{-Rfl})$$

$$\frac{(\alpha \leqslant \tau) \in \Gamma \qquad \Gamma \vdash \tau' \ <: \ \tau}{\Gamma \vdash \alpha \ <: \ \tau} \quad (S^N\text{-Var})$$

$$\frac{\Gamma \vdash \tau_2 \ <: \ \tau_1 \qquad \Gamma \vdash \tau_1' \ <: \ \tau_2'}{\Gamma \vdash \tau_1 \to \tau_1' \ <: \ \tau_2 \to \tau_2'} \quad (S^N\text{-Arr})$$

$$\frac{\Gamma \vdash \tau_2 \ <: \ \tau_1 \qquad \Gamma, (\alpha \leqslant \tau_2) \vdash \tau_1' \ <: \ \tau_2'}{\Gamma \vdash \forall(\alpha \leqslant \tau_1).\tau_1' \ <: \ \forall(\alpha \leqslant \tau_2).\tau_2'} \quad (S^N\text{-All})$$

Figure 2. System $F_{<:}$ Subtyping

System $F_{<:}$

$$\Gamma \vdash \tau \;<:\; \top \quad (S^N\text{-Top}) \qquad \Gamma \vdash \alpha \;<:\; \alpha \quad (S^N\text{-Rfl})$$

$$\frac{(\alpha \leqslant \tau) \in \Gamma \qquad \Gamma \vdash \tau' \;<:\; \tau}{\Gamma \vdash \alpha \;<:\; \tau} \quad (S^N\text{-Var})$$

$$\frac{\Gamma \vdash \tau_2 \;<:\; \tau_1 \qquad \Gamma \vdash \tau_1' \;<:\; \tau_2'}{\Gamma \vdash \tau_1 \to \tau_1' \;<:\; \tau_2 \to \tau_2'} \quad (S^N\text{-Arr})$$

$$\frac{\Gamma \vdash \tau_2 \;<:\; \tau_1 \qquad \Gamma, (\alpha \leqslant \tau_2) \vdash \tau_1' \;<:\; \tau_2'}{\Gamma \vdash \forall(\alpha \leqslant \tau_1).\tau_1' \;<:\; \forall(\alpha \leqslant \tau_2).\tau_2'} \quad (S^N\text{-All})$$

UNDECIDABLE

Figure 2. System $F_{<:}$ Subtyping

System $F_{<:}$

Undecidability is a problem for type checkers

System $F_{<:}$

Undecidability is a problem for type checkers

Gives us bad error messages when things fail

System $F_{<:}$

Basic idea:

System $F_{<:}$

Basic idea:

- We want to define a syntactic subset of System F<: that is

System $F_{<:}$

Basic idea:

- We want to define a syntactic subset of System F<: that is
  - decidable and

System $F_{<:}$

Basic idea:

- We want to define a syntactic subset of System F<: that is
  - decidable and
  - expressive enough for common programming idioms

System $F_{<:}$

Basic idea:

- We want to define a syntactic subset of System F<: that is
  - decidable and
  - expressive enough for common programming idioms
- Syntactic because

System $F_{<:}$

Basic idea:

- We want to define a syntactic subset of System F<: that is
  - decidable and
  - expressive enough for common programming idioms
- Syntactic because
  - it is easy to check

System $F_{<:}$

Basic idea:

- We want to define a syntactic subset of System F<: that is
  - decidable and
  - expressive enough for common programming idioms
- Syntactic because
  - it is easy to check
  - and retains nice System F<: properties: type safety, transitivity, etc.

System $F_{<:}$

Basic idea:

- We want to define a syntactic subset of System F<: that is
  - decidable and
  - expressive enough for common programming idioms
- Syntactic because
  - it is easy to check
  - and retains nice System F<: properties: type safety, transitivity, etc.
- What is the largest such syntactic subset?

System $F_{<:}$

Basic idea:

- We want to define a syntactic subset of System F<: that is
  - decidable and
  - expressive enough for common programming idioms
- Syntactic because
  - it is easy to check
  - and retains nice System F<: properties: type safety, transitivity, etc.
- What is the largest such syntactic subset?
- Does this subset exclude important design patterns?

System $F_{<:}$

Simplest Fix:

		Separated System $F_{<:}$ Type
τ	::=	
	$\top$	*top*
	α	*variable*
	$\tau \to \tau$	*arrow*
	$\forall(\alpha \leqslant \mu).\mu$	*restricted all*
μ	::=	**Restricted Type**
	$\top$	*top*
	γ	*variable*
	$\mu \to \mu$	*arrow*

Figure 3. A Simple Separated System $F_{<:}$ Type Syntax

System $F_{<:}^{R}$

τ	::=	**Separated System $F_{<:}$ Type**
	$\top$	*top*
	α	*variable*
	γ	*restricted variable*
	$\tau \to \tau$	*arrow*
	$\forall(\gamma \leqslant \mu).\tau$	*restricted all*
	$\forall(\delta \leqslant \tau).\tau$	*all*
α	::=	**Type Variable**
	γ	*unrestricted*
	δ	*restricted*
μ	::=	**Restricted Type**
	$\top$	*top*
	γ	*variable*
	$\mu \to \mu$	*arrow*

Figure 3. A Separated System $F_{<:}$ Type Syntax

$$\frac{\Gamma, (\alpha \leqslant \tau) \vdash \tau_1 \ <:^R \ \tau_2}{\Gamma \vdash \forall(\alpha \leqslant \tau).\tau_1 \ <:^R \ \forall(\alpha \leqslant \tau).\tau_2} \quad (S^R\text{-All-Kernel})$$

$$\frac{\Gamma \vdash \mu_2 \ <:^R \ \mu_1 \qquad \Gamma, (\gamma \leqslant \mu_2) \vdash \tau_1 \ <:^R \ \tau_2}{\Gamma \vdash \forall(\gamma \leqslant \mu_1).\tau_1 \ <:^R \ \forall(\gamma \leqslant \mu_2).\tau_2} \quad (S^R\text{-All})$$

Figure 4. Separated System $F_{<:}$ Subtyping

System $F_{<:}^{R}$

τ	::=	**Separated System $F_{<:}$ Type**
	$\top$	*top*
	α	*variable*
	γ	*restricted variable*
	$\tau \to \tau$	*arrow*
	$\forall(\gamma \leqslant \mu).\tau$	*restricted all*
	$\forall(\delta \leqslant \tau).\tau$	*all*
α	::=	**Type Variable**
	γ	*unrestricted*
	δ	*restricted*
μ	::=	**Restricted Type**
	$\top$	*top*
	γ	*variable*
	$\mu \to \mu$	*arrow*
	$\forall(\gamma \leqslant \top).\mu$	*all???*

Figure 3. A Separated System $F_{<:}$ Type Syntax

$$\frac{\Gamma, (\alpha \leqslant \tau) \vdash \tau_1 \ <:^R \ \tau_2}{\Gamma \vdash \forall(\alpha \leqslant \tau).\tau_1 \ <:^R \ \forall(\alpha \leqslant \tau).\tau_2} \quad (S^R\text{-All-Kernel})$$

$$\frac{\Gamma \vdash \mu_2 \ <:^R \ \mu_1 \qquad \Gamma, (\gamma \leqslant \mu_2) \vdash \tau_1 \ <:^R \ \tau_2}{\Gamma \vdash \forall(\gamma \leqslant \mu_1).\tau_1 \ <:^R \ \forall(\gamma \leqslant \mu_2).\tau_2} \quad (S^R\text{-All})$$

Figure 4. Separated System $F_{<:}$ Subtyping

System $F_{<:}$

Other Solutions:

System $F_{<:}$

Other Solutions:

1. Kernel $F_{<:}$ (see TAPL)

System $F_{<:}$

Other Solutions:

1. Kernel $F_{<:}$ (see TAPL)
2. $F_{<:}^{T}$

System $F_{<:}$

Other Solutions:

1. Kernel $F_{<:}$ (see TAPL)
2. $F_{<:}^{T}$
3. Strong Kernel $F_{<:}$ / Wyv_{fix}

Wyvern, Scala, and $D_{<:}$

τ	::=	$\mathbf{D_{<:}}$ **Type**
	$\top$	*top*
	$\bot$	*bottom*
	$\{L : \tau \ldots \tau\}$	*declaration*
	$x.L$	*selection*
	$\forall(x : \tau).\tau^x$	*function*

Figure 5. $D_{<:}$ Type Syntax

Wyvern, Scala, and $D_{<:}$

$$\Gamma \vdash \tau \ <: \ \top \quad \text{(Top)} \qquad \Gamma \vdash \tau \ <: \ \tau \quad \text{(Rfl)}$$

$$\frac{\Gamma(x) = \{L \ : \tau_1 \ldots \tau_2\}}{\Gamma \vdash x.L \ <: \ \tau_2} \ \text{(Sel1)} \qquad \frac{\Gamma(x) = \{L \ : \tau_1 \ldots \tau_2\}}{\Gamma \vdash \tau_1 \ <: \ x.L} \ \text{(Sel2)}$$

$$\frac{\Gamma \vdash \tau_1 \ <: \ \tau_2 \qquad \Gamma \vdash \tau_2 \ <: \ \tau_3}{\Gamma \vdash \tau_1 \ <: \ \tau_3} \ \text{(Trans)}$$

$$\frac{\Gamma \vdash \tau_2 \ <: \ \tau_1 \qquad \Gamma \vdash \tau_1' \ <: \ \tau_2'}{\Gamma \vdash \{L \ : \tau_1 \ldots \tau_1'\} \ <: \ \{L \ : \tau_2 \ldots \tau_2'\}} \ \text{(Bnd)}$$

$$\frac{\Gamma \vdash \tau_2 \ <: \ \tau_1 \qquad \Gamma, (x : \tau_2) \vdash \tau_1' \ <: \ \tau_2'}{\Gamma \vdash \forall (x : \tau_1).\tau_1' \ <: \ \forall (x : \tau_2).\tau_2'} \ \text{(All)}$$

Figure 6. $D_{<:}$ Subtyping

Wyvern, Scala, and $D_{<:}$

$$\Gamma \vdash \tau \ <: \ \top \quad \text{(Top)} \qquad \Gamma \vdash \tau \ <: \ \tau \quad \text{(Rfl)}$$

$$\frac{\Gamma(x) = \{L \ : \tau_1 \ldots \tau_2\}}{\Gamma \vdash x.L \ <: \ \tau_2} \ \text{(Sel1)} \qquad \frac{\Gamma(x) = \{L \ : \tau_1 \ldots \tau_2\}}{\Gamma \vdash \tau_1 \ <: \ x.L} \ \text{(Sel2)}$$

$$\frac{\Gamma \vdash \tau_1 \ <: \ \tau_2 \qquad \Gamma \vdash \tau_2 \ <: \ \tau_3}{\Gamma \vdash \tau_1 \ <: \ \tau_3} \ \text{(Trans)}$$

$$\frac{\Gamma \vdash \tau_2 \ <: \ \tau_1 \qquad \Gamma \vdash \tau_1' \ <: \ \tau_2'}{\Gamma \vdash \{L \ : \tau_1 \ldots \tau_1'\} \ <: \ \{L \ : \tau_2 \ldots \tau_2'\}} \ \text{(Bnd)}$$

$$\frac{\Gamma \vdash \tau_2 \ <: \ \tau_1 \qquad \Gamma, (x : \tau_2) \vdash \tau_1' \ <: \ \tau_2'}{\Gamma \vdash \forall (x : \tau_1).\tau_1' \ <: \ \forall (x : \tau_2).\tau_2'} \ \text{(All)}$$

Figure 6. $D_{<:}$ Subtyping

Wyvern, Scala, and $D_{<:}$

$$\Gamma \vdash \tau \ <: \ \top \quad \text{(Top)} \qquad \Gamma \vdash \tau \ <: \ \tau \quad \text{(Rfl)}$$

$$\frac{\Gamma(x) = \{L \ : \tau_1 \ldots \tau_2\}}{\Gamma \vdash x.L \ <: \ \tau_2} \quad \text{(Sel1)} \qquad \frac{\Gamma(x) = \{L \ : \tau_1 \ldots \tau_2\}}{\Gamma \vdash \tau_1 \ <: \ x.L} \quad \text{(Sel2)}$$

$$\frac{\Gamma \vdash \tau_1 \ <: \ \tau_2 \qquad \Gamma \vdash \tau_2 \ <: \ \tau_3}{\Gamma \vdash \tau_1 \ <: \ \tau_3} \quad \text{(Trans)}$$

$$\frac{\Gamma \vdash \tau_2 \ <: \ \tau_1 \qquad \Gamma \vdash \tau_1' \ <: \ \tau_2'}{\Gamma \vdash \{L \ : \tau_1 \ldots \tau_1'\} \ <: \ \{L \ : \tau_2 \ldots \tau_2'\}} \quad \text{(Bnd)}$$

$$\frac{\Gamma \vdash \tau_2 \ <: \ \tau_1 \qquad \Gamma, (x : \tau_2) \vdash \tau_1' \ <: \ \tau_2'}{\Gamma \vdash \forall (x : \tau_1).\tau_1' \ <: \ \forall (x : \tau_2).\tau_2'} \quad \text{(All)}$$

Figure 6. $D_{<:}$ Subtyping

Wyvern, Scala, and $D_{<:}$

- recursive types?
- intersection and union types?
- a survey of bounded quantification in real code
  - is there a corpus we could use?