Polarized Higher-Order Subtyping

Martin Steffen

15. April 1999

Aalborg

Overview

- Motivation
- Types & object-orientation
- “F-omega-sub”
- Decidability
- Conclusion

Typed programming

- "semantical" phase of compilers:

lexer ⇝	parser ⇝	type system ⇝	"oper. semantics"
regular	context-free	?	undecidable

- strong type safety:

  well-typed programs are free of run-time errors

- preferably: statically checkable ($\Rightarrow$ efficiency)

Parametric polymorphism

- polymorphic: a program can carry more than one type
- Example: swapping arguments

$$swap(x{:}\mathbb{N}, y{:}\mathbb{B}) \quad = \quad (y, x) \quad : \quad \mathbb{N} \times \mathbb{B} \rightarrow \mathbb{B} \times \mathbb{N}$$

- preferable: one generic *swap*-function for all types

$$swap : \forall X, Y. X \times Y \rightarrow Y \times X$$
$$swap\ X\ Y\ (x{:}X, y{:}Y) = (y, x)$$

⇒ **parametric/universal** polymorphism

Subtyping

Is S a **subtype** of T, then a program of type S can be safely used in place where a program of type T is expected

⇒ order on the types ($\leq$)

- intuitively: subsets $S \subseteq T$, e.g. $Int \leq Real$.
- combination with universal polymorphism:

$$list\_max : \forall X \leq Ord.(List\ of\ X) \to X$$

- bounded universal quantification

Type operators

- Example 1: "$\boldsymbol{List\ of}$ _" is no type, only "$\boldsymbol{List\ of}\ \mathbb{N}$" is, e.g.

$$[4, 5, 0] : \boldsymbol{List\ of}\ \mathbb{N}$$

⇒ type **operator** = function from types to types

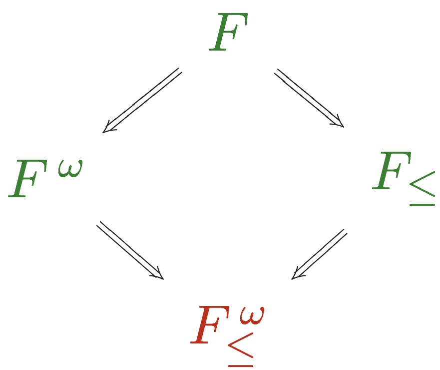
- Example 2: signature/method interface of objects

$$\begin{aligned} PointSig \quad of \;\; X = \{ \;\; & getx : X \rightarrow \mathbb{N}, \\ & setx : X \rightarrow \mathbb{N} \rightarrow X \; \} \end{aligned}$$

Features (cont.)

- polymorphism
  - universal polymorphism
  - subtyping
- higher-order functions
- encapsulation
- inheritance
- late/dynamic binding
- ...

Formal model: typed λ-calculi



- F: the polymorphic λ-calculus [Girard, 1971] [Reynolds, 1974]
- $F_{\leq}$: [Cardelli and Wegner, 1985] ...
- F^{ω} [Girard, 1971]
- $F^{\omega}_{\leq}$ [Cardelli, 1990] [Mitchell, 1990] ...

$F^{\omega}_{\leq}$ as OO-calculus

- [Hofmann and Pierce, 1995]: $F^{\omega}_{\leq}$ as base calculus for OO-languages
  - class-based
  - single inheritance
  - encapsulation (using $\exists$)

  provided:

  signature/method interface = monotone type operator

$$PointSig = Fun(X).\{\ getx : X \to \mathbb{N},\ setx : X \to \mathbb{N} \to \boldsymbol{X}\ \}$$

- class-inheritance $\Rightarrow$ subtype relation between instances
- absence of binary methods
- "inheritance of proofs" [Hofmann et al., 1998]

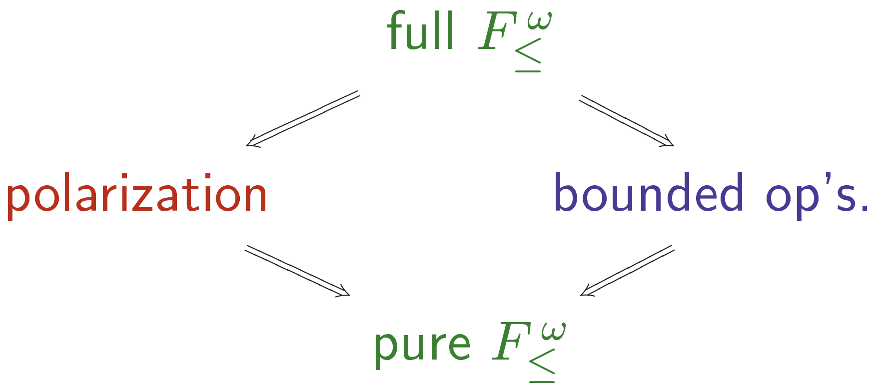
Example: more flexible typing

- Cf. [Duggan and Compagnoni, 1999] (for object type constructors)
- Example:

If $\quad Int \leq Real,$

then $\quad \boldsymbol{Array\ of}\ Int \leq \boldsymbol{Array\ of}\ Real\ ?$

$F^{\omega}_{\leq}$ = "the" calculus of higher-order subtyping?



- full $F^{\omega}_{\leq}$: [Cardelli, 1990]
- bounded operator abstraction: [Compagnoni and Goguen, 1997]

What's next

- fix the syntax
- axiomatize static properties
  - when does program t carries type T?
  - when is type S a subtype of type T?

$\Rightarrow$ formal deduction system

Syntax of $F^\omega_\leq$

- three levels: programs, types and kinds

t	$::=$	$x \mid fun(x{:}T)t \mid t\ t$	functions
	$\mid$	$fun(X \leq T)t \mid t\ T$	universal polymorphism, $\leq$
T	$::=$	$T \to T$	functions
	$\mid$	$All(X \leq T)T \mid X$	universal polymorphism, $\leq$
	$\mid$	$Top(K)$	
	$\mid$	$Fun(X{:}K)T \mid T\ T$	type operators
K	$::=$	$\star \mid K \to K$	kinds (= "type of types")

Judgments & rules

- So far: syntax only (context-free), but no relationships

⇒ Judgments (e.g.):

$\Gamma \vdash t : T$	program t is of type T
$\Gamma \vdash S \leq T$	S is a subtype of T
$\Gamma \vdash T : K$	type T is of kind K

- Dependency: Subsumption

$$\frac{\Gamma \vdash \boldsymbol{t : S} \qquad \Gamma \vdash \boldsymbol{S \leq T}}{\Gamma \vdash \boldsymbol{t : T}} \qquad (\textsc{Sub})$$

$F^{\omega}_{\leq}$: subtype system

- axiomatization of $\leq$ by deduction rules
- two classes of $\leq$-rules

1. language-independent properties of $\leq$, e.g. transitivity.

$$\frac{\Gamma \vdash \boldsymbol{S} \leq \boldsymbol{U} \quad \Gamma \vdash \boldsymbol{U} \leq \boldsymbol{T}}{\Gamma \vdash \boldsymbol{S} \leq \boldsymbol{T}} \tag{S-Trans}$$

2. structural, e.g. for $\rightarrow$–types

$$\frac{\Gamma \vdash \boldsymbol{T_1} \leq \boldsymbol{S_1} \qquad \Gamma \vdash \boldsymbol{S_2} \leq \boldsymbol{T_2}}{\Gamma \vdash \boldsymbol{S_1} \rightarrow \boldsymbol{S_2} \leq \boldsymbol{T_1} \rightarrow \boldsymbol{T_2}} \tag{S-Arrow}$$

$F^{\omega}_{\leq}$ + monotonicity

- extension of $F^{\omega}_{\leq}$ by monotonicity information ⇒

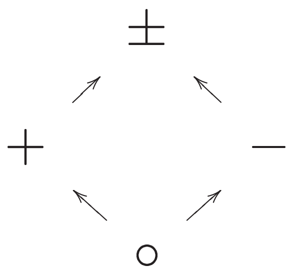
"polarized $F^{\omega}_{\leq}$"

If $\quad Int \leq Real,$

then $\quad \boldsymbol{List\ of}\ Int \leq \boldsymbol{List\ of}\ Real?$

$$\frac{\ldots \boldsymbol{X_1 \leq X_2} \ldots \vdash T\ X_1 \leq T\ X_2}{\Gamma \vdash T \in K_1 \rightarrow^{+} K_2}$$

- all in all: 4 polarities ⇒ subkinding



Goal

- Given: specification of the (sub–)type systems
- Needed: algorithm to check the judgments.

$$\text{type system} \quad \overset{?}{\Longrightarrow} \quad \text{type checker}$$

Where is the problem, then?

structural rules $\rightarrow$, $\forall$ ... straightforward (or almost ...)

$$\frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma \vdash S_2 \leq T_2}{\Gamma \vdash S_1 \rightarrow S_2 \leq T_1 \rightarrow T_2}$$

but not

1. transitivity:

$$\frac{\Gamma \vdash S \leq U \quad \Gamma \vdash U \leq T}{\Gamma \vdash S \leq T}$$

2. conversion

$$\frac{S =_\beta S' \qquad \Gamma \vdash S' \leq T' \qquad T' =_\beta T}{\Gamma \vdash S \leq T}$$

Transitivity

- Goal: rule of transitivity is superfluous = "cut elimination"

$$\frac{U_1 \le S_1 \quad S_2 \le U_2}{S_1 \to S_2 \le U_1 \to U_2} + \frac{T_1 \le U_1 \quad U_2 \le S_2}{U_1 \to U_2 \le T_1 \to T_2}$$

$$\frac{\dfrac{T_1 \le U_1 \quad U_1 \le S_1}{T_1 \le S_1} \qquad \dfrac{S_2 \le U_2 \quad U_2 \le T_2}{S_2 \le T_2}}{S_1 \to S_2 \le T_1 \to T_2}$$

- Problems:
  - S-TRANS is not superfluous (known twist)
  - destroys the normal form

Eliminate cut?

- alas, $\textsc{S-Trans}$ is not superfluous:

- example:[1] assume $\Gamma = \ldots X \leq Y, Y \leq Z \ldots.$

$$\frac{\Gamma \vdash X \leq Y \qquad \Gamma \vdash Y \leq Z}{\Gamma \vdash X \leq Z}$$

- solution: add a new rule

$$\frac{\Gamma \vdash \Gamma(X) \leq T}{\Gamma \vdash X \leq T}$$

[1] one can have more complicated examples in F^{ω}

Conversion

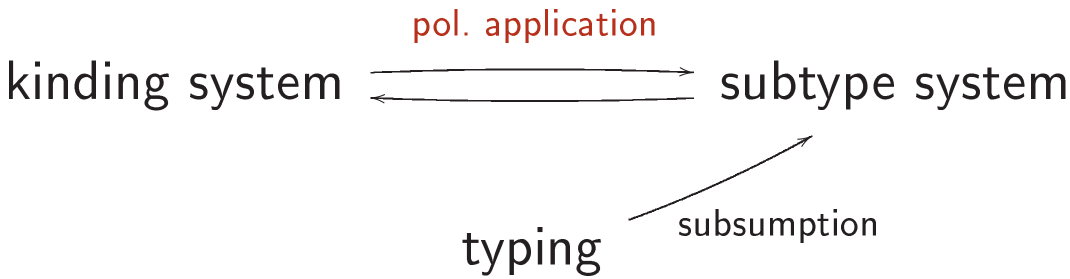
- Goal: Using normal forms only
- instead of undirected conversion: reduction

$$\frac{S =_\beta S' \qquad T =_\beta T' \qquad \Gamma \vdash S' \leq T'}{\Gamma \vdash S \leq T}$$

- For instance: for arrow-types

$$\frac{S \rightarrow^*_{\beta\top} S_1 \rightarrow S_2 \qquad T \rightarrow^*_{\beta\top} T_1 \rightarrow T_2 \qquad \Gamma \vdash T_1 \leq S_1 \in \star \qquad \Gamma \vdash S_2 \leq T_2 \in \star}{\Gamma \vdash S \leq T \in \star}$$

Additional problems



$$\frac{\begin{array}{c}\cdots\\ \ldots \boldsymbol{X_1} \leq \boldsymbol{X_2} \ldots \vdash T\ X_1 \leq T\ X_2\end{array}}{\Gamma \vdash T \in K_1 \rightarrow^{+} K_2}$$

- break the direct interdependence of subtyping and kinding $\Rightarrow$ "stratification"
- termination
- generalization of $\forall$-subtyping rule
- "antisymmetry" of $\leq$ (cf. [Compagnoni and Goguen, 1999])

Results

Theorem. *Subtyping* $\Gamma \vdash S \leq T : K$ *and kinding* $\Gamma \vdash T : K$ *for polarized* $F^{\omega}_{\leq}$ *are decidable.*

Proposition. *Every well-typed program has a minimal type.*

Corollary. *Typing* $\Gamma \vdash t : T$ *for polarized* $F^{\omega}_{\leq}$ *is decidable.*

Future work

- Model (for instance PER-model)
- decidability for the full calculus ([Compagnoni and Goguen, 1997]):

$$Fun(X \leq S)T \qquad \text{instead of} \qquad Fun(X : K)T$$

- local type inference (e.g. [Pierce and Turner, 1998] for $F_{\leq}$, in Pict)

References

[Cardelli, 1990] Cardelli, L. (1990). Notes about $F^{\omega}_{<:}$. Unpublished manuscript.

[Cardelli and Wegner, 1985] Cardelli, L. and Wegner, P. (1985). On understanding types, data abstraction and polymorphism. *Computing Surveys*, 17(4):471–522.

[Compagnoni and Goguen, 1997] Compagnoni, A. and Goguen, H. (1997). Typed operational semantics for higher order subtyping. Technical Report ECS-LFCS-97-361, Department of Computer Science, University of Edinburgh. Submitted for publication in *Information and Computation*.

[Compagnoni and Goguen, 1999] Compagnoni, A. and Goguen, H. (1999). Antisymmetry for higher-order subtyping. submitted for publication.

[Duggan and Compagnoni, 1999] Duggan, D. and Compagnoni, A. (1999). Flexible subtyping with object type constructors. submitted.

[Girard, 1971] Girard, J.-Y. (1971). Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In Fenstad, J. E., editor, *Second Scandinavian Logic Symposium '71 (Oslo, Norway)*, number 63 in Studies in Logic and the Foundations of Mathematics, pages 63–92. North-Holland.

[Gunter and Mitchell, 1994] Gunter, C. A. and Mitchell, J. C. (1994). *Theoretical Aspects of Object-Oriented Programming, Types, Semantics, and Language Design*. Foundations of Computing Series. MIT Press.

[Hofmann et al., 1998] Hofmann, M., Naraschewski, W., Steffen, M., and Stroup, T. (1998). Inheritance of proofs. *Theory and Practice of Object Systems (Tapos), Special Issue on Third Workshop on Foundations of Object-Oriented Languages (FOOL 3), July 1996*, 4(1):51–69. An extended version appeared as Interner Bericht, Universität Erlangen-Nürnberg, IMMDVII-5/96.

[Hofmann and Pierce, 1995] Hofmann, M. and Pierce, B. (1995). A unifying type-theoretic framework for objects. *Journal of Functional Programming*, 5(4):593–635. Previous versions appeared in the Symposium on Theoretical Aspects of Computer Science, 1994, (pages 251–262) and, under the title "An Abstract View of Objects and Subtyping (Preliminary Report)," as University of Edinburgh, LFCS technical report ECS-LFCS-92-226, 1992.

[Mitchell, 1990] Mitchell, J. C. (1990). Toward a typed foundation for method specialization and inheritance.

In *Seventeenth Annual Symposium on Principles of Programming Languages (POPL) (San Fancisco, CA)*, pages 109–124. ACM. Also in the collection [?].

[Pierce and Turner, 1998] Pierce, B. C. and Turner, D. N. (1998). Local type inference. In *Proceedings of POPL '98*. ACM. Also as Indiana University Technical Report CSCI TR #493.

[Reynolds, 1974] Reynolds, J. (1974). Towards a theory of type structure. In Robinet, B., editor, *Colloque sur la programmation (Paris, France)*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer-Verlag.