Algebraic Subtyping for Algebraic Effects and Handlers

Axel Faes   promotor: Tom Schrijvers   advisor: Amr Hany Saleh

Introduction

Algebraic effect handlers
A feature for side effects and exception handlers on steroids [1, 4, 5]

Implemented in Eff programming language [6, 7]

Algebraic subtyping
A form of subtyping used in type inference systems [2]

```
let twice f x =
    f (f x)
```

```
effect Op : unit -> int

let someFun b =
    handle (
        if (b == 0) then
            let a = #Op ()
            print a
        else
            print b
    ) with
        | #Op () cont -> cont 1
```

With subtyping
twice : (β -> γ) -> α -> γ | α ≤ β, γ ≤ β

With algebraic subtyping
twice : (α -> α & β) -> α -> β

Research problem

Too many constraints
Constraints keep stacking and they become unwieldy.

Algebraic Subtyping
Does not use effect information.

Optimisations
Implementing effect-aware optimizing compiler is error-prone due to the many different constraints that remain unsolved in traditional subtyping systems.

How can effects be introduced within algebraic subtyping? How can such a system be implemented in the Eff programming language?

Extension

value v	$::=$	x	λ-variable
	$\mid$	$\hat{\mathbf{x}}$	let-variable
	$\mid$	`true`	true
	$\mid$	`false`	false
	$\mid$	$\lambda x.c$	function
	$\mid$	$\{$	**handler**
		`return` $x \mapsto c_r$,	**return case**
		$[\mathsf{Op}\ y\ k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}$	**operation cases**
		$\}$	
comp c	$::=$	$v_1\ v_2$	application
	$\mid$	`do` $\hat{\mathbf{x}} = c_1$; c_2	sequencing
	$\mid$	`let` $\hat{\mathbf{x}} = v$ `in` c	let
	$\mid$	`if` e `then` c_1 `else` c_2	conditional
	$\mid$	`return` v	**returned val**
	$\mid$	`Op` v	**operation call**
	$\mid$	`handle` c `with` v	**handling**

(pure) type A, B	$::=$	bool	bool type
	$\mid$	$A \to \underline{C}$	function type
	$\mid$	$\underline{C} \Rightarrow \underline{D}$	**handler type**
	$\mid$	α	type variable
	$\mid$	$\mu\alpha.A$	recursive type
	$\mid$	$\top$	top
	$\mid$	$\bot$	bottom
	$\mid$	$A \sqcap B$	intersection
	$\mid$	$A \sqcup B$	union
dirty type $\underline{C}, \underline{D}$	$::=$	$A \,!\, \Delta$	
dirt Δ	$::=$	Op	operation
	$\mid$	δ	dirt variable
	$\mid$	$\emptyset$	empty dirt
	$\mid$	$\Delta_1 \sqcap \Delta_2$	intersection
	$\mid$	$\Delta_1 \sqcup \Delta_2$	union

$$(A_1 \to \underline{C}_1) \sqcup (A_2 \to \underline{C}_2) \equiv (A_1 \sqcap A_2) \to (\underline{C}_1 \sqcup \underline{C}_2)$$

Formulation of typing rules

Relationship to subtyping

Biunification algorithm
    input <-> output

Type inference algorithm

Type simplification algorithm
    Construct type automata
    Convert to NFA
    Minimization algorithm

Semantics of the dirt
Set operations?
(Op ⊔ Op2) ⊓ (Op ⊔ Op3)
    => Op

Input vs Output
    ⊔ for outputs
    ⊓ for inputs

$$\Delta_1 \leqslant \Delta_2 \leftrightarrow \Delta_1 \sqcup \Delta_2 \equiv \Delta_2$$

$$\Delta_1 \leqslant \Delta_2 \leftrightarrow \Delta_1 \equiv \Delta_1 \sqcap \Delta_2$$

Evaluation

Implementation
Eff programming language
Fully featured

Replace type inference engine
    ~2900 loc ⇔ ~5800

Proofs
Algorithms have been proven to be correct (see thesis appendix for the proofs)

Program \ System	Algebraic Subtyping	Standard Subtyping
Interpreter	**1.550**	1.740
Loop	**1.285**	2.859
Parser	171.887	**11.791**
N-Queens	35.377	**5.362**
Range	**0.642**	1.058

Future Work

Optimizations engine
Adapting the effect-aware optimizing compiler for the algebraic-subtyping based system.

Simplification of types and effects
Implementing the extended algorithm to simplify types and effects using type automata.

Summary

Algebraic effects and handlers
These are a very active area of research. An important aspect is the development of an optimising compiler.

Research problem
Compilation is a slow process with difficult to read types due to the constraints without a strong, reliable type-&-effect system.

This thesis simplifies constraint generation for types ***AND EFFECTS*** by providing a new core language based upon extended **algebraic subtyping.**

References

[1] Andrej Bauer and Matija Pretnar. 2014. An Effect System for Algebraic Effects and Handlers. Logical Methods in Computer Science 10, 4 (2014). https://doi.org/10.2168/LMCS-10(4:9)2014

[2] Stephen Dolan and Alan Mycroft. 2017. Polymorphism, Subtyping, and Type Inference in MLsub. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017). ACM, New York, NY, USA, 60–72. https://doi.org/10.1145/3009837.3009882

[3] Sam Lindley, Conor McBride, and Craig McLaughlin. 2017. Do Be Do Be Do. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017). ACM, New York, NY, USA, 500–514. https://doi.org/10.1145/3009837.3009897

[4] Gordon D. Plotkin and Matija Pretnar. 2013. Handling Algebraic Effects. Logical Methods in Computer Science 9, 4 (2013). https://doi.org/10.2168/LMCS-9(4:23)2013

[5] Matija Pretnar. 2014. Inferring Algebraic Effects. Logical Methods in Computer Science 10, 3 (2014). https://doi.org/10. 2168/LMCS-10(3:21)2014

[6] Matija Pretnar. 2015. An introduction to algebraic effects and handlers. invited tutorial paper. Electronic Notes in Theoretical Computer Science 319 (2015), 19–35.

[7] Andrej Bauer and Matija Pretnar. 2015. Programming with algebraic effects and handlers. J. Log. Algebr. Meth. Program. 84, 1 (2015), 108–123. https://doi.org/10.1016/j.jlamp.2014.02.001

Acknowledgements

I would like to thank Amr Hany Saleh for his continuous guidance and help. I would also like to thank Matija Pretnar and Tom Schrijvers for their support during my research.

KU LEUVEN

DTAI
Declaratieve Talen en Artificiële Intelligentie