KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Algebraic Subtyping for Algebraic Types and Effects

Axel Faes

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotor:
Prof. dr. ir. Tom Schrijvers

Assessor:
assesors

Begeleider:
Amr Hany Saleh

Academiejaar 2017 – 2018

Voorwoord

I would like to thank everybody who kept me busy the last year, especially my promoter and my assistants. I would also like to thank the jury for reading the text.

Axel Faes

Inhoudsopgave

Samenvatting

Algebraic effects and handlers are a very active area of research. An important aspect is the development of an optimising compiler. Eff is an ML-style language with support for effects and forms the testbed for the optimising compiler. However, the type-&-effect system of Eff is unsatisfactory. This is due to the lack of some elegant properties. It is also awkward to implement and use in practice.

Lijst van figuren

Lijst van tabellen

Hoofdstuk 1

Introduction

The specification for a type-&-effect system with algebraic subtyping for algebraic effects and handlers is given in this document. The formal properties of this system are studied in order to find which properties are satisfied compared to other type-&-effect systems. The proposed type-&-effect system builds on two very recent developments in the area of programming language theory.

Algebraic subtyping

In his December 2016 PhD thesis, Stephen Dolan (University of Cambridge, UK), has presented a novel type system that combines subtyping and parametric polymorphism in a particulary attractive and elegant fashion. A cornerstone of his design are the algebraic properties that the subtyping relation should respect.

Algebraic effects and handlers

These are a new formalism for formally modelling side-effects (e.g. mutable state or non-determinism) in programming languages, developed by Matija Pretnar (University of Ljubjana) and Gordon Plotkin (University of Edinburgh). This approach is gaining a lot of traction, not only as a formalism but also as a practical feature in actual programming languages (e.g. the Koka language developed by Microsoft Research). We are collaborating with Matija Pretnar on the efficient implementation of one such language, called Eff. Axel Faes has contributed to this collaboration during a project he did for the Honoursprogramme of the Faculty of Engineering Science.

1.1 Motivation

Algebraic effects and handlers benefit from a custom type-&-effect system, a type system that also tracks which effects can happen in a program. Several such type-&-effect systems have been proposed in the literature, but all are unsatisfactory. We attribute this to the lack of the elegant properties of Dolan's type system. Indeed the existing type-&-effect systems are not only theoretically unsatisfactory, but they are also awkward to implement and use in practice.

Research questions

- How can Dolan's elegant type system be extended with effect information?
- Which properties are preserved and which aren't preserved?
- What advantages are there to an type-&-effect system based on Dolan's elegant type system?

1.2 Goals

The goal of this thesis is to derive a type-&-effect system that extends Dolan's elegant type system with effect information. This type-&-effect system should inherit Dolan's harmonious combination of subtyping (in our case induced by a lattice structure on the effect information) with parametric polymorphism and preserve all of its desirable properties (both low-level algebraic properties and high-level meta-theoretical properties like type soundness and the existence of principal types). Afterwards this type-&-effect system The following approach is taken:

1. Study of the relevant literature and theoretical background.
2. Design of a type-&-effect system derived from Dolan's, that integrates effects.
3. Proving the desirable properties of the proposed type-&-effect system: type soundness, principal typing, ...
4. Time permitting: Design of a type inference algorithm that derives the principal types of programs without type annotations and proving its correctness.
5. Time permitting: Implementation of the algorithm and comparing it to other algorithms (such as row polymorphism based type-&-effect systems).

1.3 Results

Describe what the resulting product is and how it is useful or provides an advantage over other solutions.

Hoofdstuk 2

Background

In this section, I will provide the background necessary to be able to read the text. This includes an introduction into programming languages (and programming language theory) and algebraic effect handlers

Dolan's type system and EFF are discussed in further chapters and thus shouldn't need to be explained in this section.

Hoofdstuk 3

Related Work (Algebraic Subtyping)

Subtyping is a partial order which is a reflexive transitive binary relation satisfying antisymmetry (subtyping rules). The subtyping order also forms a distributive lattice (equivalence rules).

Hoofdstuk 4

Related Work (Eff)

The type-&-effect system that is used in EFF is based on subtyping and dirty types [1].

4.1 Types and terms

Terms

Figure 4.1 shows the two types of terms in EFF. There are values v and computations c. Computations are terms that can contain effects. Effects are denoted as operations Op which can be called.

value v	$::=$	x	variable
	|	`true`	true
	|	`false`	false
	|	$\lambda x.c$	function
	|	$\{$	handler
		$\quad$ `return` $x \mapsto c_r,$	return case
		$\quad [\mathtt{Op}\ y\ k \mapsto c_{\mathtt{Op}}]_{\mathtt{Op} \in O}$	operation cases
		$\}$	
comp c	$::=$	$v_1\ v_2$	application
	|	`let rec` $f\ x = c_1$ `in` c_2	rec definition
	|	`return` v	returned val
	|	$\mathtt{Op}\ v$	operation call
	|	`do` $x \leftarrow c_1\ ;\ c_2$	sequencing
	|	`handle` c `with` v	handling

Figuur 4.1: Terms of EFF

Types

Figure 4.2 shows the types of Eff. There are two main sorts of types. There are (pure) types A, B and dirty types $\underline{C}, \underline{D}$. A dirty type is a pure type A tagged with a finite set of operations Δ, which we call dirt, that can be called. This finite set Δ is an over-approximation of the operations that are actually called. The type $\underline{C} \Rightarrow \underline{D}$ is used for handlers because a handler takes an input computation $\underline{C}$, handles the effects in this computation and outputs computation $\underline{D}$ as the result.

$$
\begin{array}{rcll}
\text{(pure) type } A, B & ::= & \texttt{bool} & \text{bool type} \\
 & | & A \to \underline{C} & \text{function type} \\
 & | & \underline{C} \Rightarrow \underline{D} & \text{handler type} \\
\text{dirty type } \underline{C}, \underline{D} & ::= & A \mathrel{!} \Delta & \\
\text{dirt } \Delta & ::= & \{\texttt{Op}_1, \ldots, \texttt{Op}_n\} &
\end{array}
$$

Figuur 4.2: Types of Eff

4.2 Type System

4.2.1 Subtyping

The dirty type $A \mathrel{!} \Delta$ is assigned to a computation returning values of type A and potentially calling operations from the set Δ. This set Δ is always an over-approximation of the actually called operations, and may safely be increased, inducing a natural subtyping judgement $A \mathrel{!} \Delta \leq A \mathrel{!} \Delta'$ on dirty types. As dirty types can occur inside pure types, we also get a derived subtyping judgement on pure types. Both judgements are defined in Figure 4.3. Observe that, as usual, subtyping is contravariant in the argument types of functions and handlers, and covariant in their return types.

Subtyping

$$
\textsc{Sub-}\texttt{bool}\quad \frac{}{\texttt{bool} \leqslant \texttt{bool}}
\qquad
\textsc{Sub-}{\to}\quad \frac{A' \leqslant A \qquad \underline{C} \leqslant \underline{C}'}{A \to \underline{C} \leqslant A' \to \underline{C}'}
\qquad
\textsc{Sub-}{\Rightarrow}\quad \frac{\underline{C}' \leqslant \underline{C} \qquad \underline{D} \leqslant \underline{D}'}{\underline{C} \Rightarrow \underline{D} \leqslant \underline{C}' \Rightarrow \underline{D}'}
$$

$$
\textsc{Sub-!}\quad \frac{A \leqslant A' \qquad \Delta \subseteq \Delta'}{A \mathrel{!} \Delta \leqslant A' \mathrel{!} \Delta'}
$$

Figuur 4.3: Subtyping for pure and dirty types of Eff

4.2.2 Typing rules

Figure 4.4 defines the typing judgements for values and computations with respect to a standard typing context Γ.

Values

The rules for subtyping, variables, and functions are entirely standard. For constants we assume a signature Σ that assigns a type A to each constant $\mathtt{k}$, which we write as $(\mathtt{k} : A) \in \Sigma$.

A handler expression has type $A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta$ iff all branches (both the operation cases and the return case) have dirty type $B \,!\, \Delta$ and the operation cases cover the set of operations O. Note that the intersection $\Delta \cap O$ is not necessarily empty. The handler deals with the operations O, but in the process may re-issue some of them (i.e., $\Delta \cap O$).

When typing operation cases, the given signature for the operation $(\mathtt{Op} : A_{\mathtt{Op}} \to B_{\mathtt{Op}}) \in \Sigma$ determines the type $A_{\mathtt{Op}}$ of the parameter x and the domain $B_{\mathtt{Op}}$ of the continuation k. As our handlers are deep, the codomain of k should be the same as the type $B \,!\, \Delta$ of the cases.

Computations

With the following exceptions, the typing judgement $\Gamma \vdash c : \underline{C}$ has a straightforward definition. The `return` construct renders a value v as a pure computation, i.e., with empty dirt. An operation invocation $\mathtt{Op}\, v$ is typed according to the operation's signature, with the operation itself as its only operation. Finally, rule $\textsc{With}$ shows that a handler with type $\underline{C} \Rightarrow \underline{D}$ transforms a computation with type $\underline{C}$ into a computation with type $\underline{D}$.

typing contexts $\Gamma \ ::= \ \epsilon \ | \ \Gamma, x : A$

Expressions

SubVal
$$\frac{\Gamma \vdash v : A \qquad A \leqslant A'}{\Gamma \vdash v : A'}$$

Var
$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$$

True
$$\frac{}{\Gamma \vdash \texttt{true} : \mathit{bool}}$$

False
$$\frac{}{\Gamma \vdash \texttt{false} : \mathit{bool}}$$

Fun
$$\frac{\Gamma, x : A \vdash c : \underline{C}}{\Gamma \vdash \lambda x.c : A \to \underline{C}}$$

Hand
$$\frac{\begin{array}{c} \Gamma, x : A \vdash c_r : B \,!\, \Delta \\ \left[(\texttt{Op} : A_{\texttt{Op}} \to B_{\texttt{Op}}) \in \Sigma \qquad \Gamma, x : A_{\texttt{Op}}, k : B_{\texttt{Op}} \to B \,!\, \Delta \vdash c_{\texttt{Op}} : B \,!\, \Delta \right]_{\texttt{Op} \in O} \end{array}}{\Gamma \vdash \{\texttt{return}\ x \mapsto c_r, [\texttt{Op}\ y\ k \mapsto c_{\texttt{Op}}]_{\texttt{Op} \in O}\} : \quad A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta}$$

Computations

SubComp
$$\frac{\Gamma \vdash c : \underline{C} \qquad \underline{C} \leqslant \underline{C}'}{\Gamma \vdash c : \underline{C}'}$$

App
$$\frac{\Gamma \vdash v_1 : A \to \underline{C} \qquad \Gamma \vdash v_2 : A}{\Gamma \vdash v_1\, v_2 : \underline{C}}$$

LetRec
$$\frac{\Gamma, f : A \to \underline{C}, x : A \vdash c_1 : \underline{C} \qquad \Gamma, f : A \to \underline{C} \vdash c_2 : \underline{D}}{\Gamma \vdash \texttt{let rec}\ f\ x = c_1\ \texttt{in}\ c_2 : \underline{D}}$$

Ret
$$\frac{\Gamma \vdash v : A}{\Gamma \vdash \texttt{return}\ v : A \,!\, \emptyset}$$

Op
$$\frac{(\texttt{Op} : A \to B) \in \Sigma \qquad \Gamma \vdash v : A}{\Gamma \vdash \texttt{Op}\, v : B \,!\, \{\texttt{Op}\}}$$

Do
$$\frac{\Gamma \vdash c_1 : A \,!\, \Delta \qquad \Gamma, x : A \vdash c_2 : B \,!\, \Delta}{\Gamma \vdash \texttt{do}\ x \leftarrow c_1 \,;\, c_2 : B \,!\, \Delta}$$

With
$$\frac{\Gamma \vdash v : \underline{C} \Rightarrow \underline{D} \qquad \Gamma \vdash c : \underline{C}}{\Gamma \vdash \texttt{handle}\ c\ \texttt{with}\ v : \underline{D}}$$

Figuur 4.4: Typing of Eff

Hoofdstuk 5

Core Language (EffCore)

EffCore is a language with row-based effects, intersection and union types and effects and is subtyping based.

Define your problem very clearly. Provide a formal definition if possible, using mathematical definitions.

5.1 Types and terms

Terms

Figure 5.1 shows the two types of terms in EffCore. There are values v and computations c. Computations are terms that can contain effects. Effects are denoted as operations Op which can be called. The function term is explicitly annotated with a type and type abstraction and type application has been added to the language. These terms only work on pure types.

Types

Figure 5.2 shows the types of EffCore. There are two main sorts of types. There are (pure) types A, B and dirty types $\underline{C}, \underline{D}$. A dirty type is a pure type A tagged with a finite set of operations Δ, which we call dirt, that can be called. It can also be an union or intersection of dirty types. In further sections, the relations between dirty intersections or unions and pure intersections or unions are explained. The finite set Δ is an over-approximation of the operations that are actually called. Row variables are introduced as well as intersection and unions. The .(DOT) is used to close rows that do not end with a row variable. The type $\underline{C} \Rightarrow \underline{D}$ is used for handlers because a handler takes an input computation $\underline{C}$, handles the effects in this computation and outputs computation $\underline{D}$ as the result.

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

Figuur 5.1: Terms of EffCore

(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	row variable
	$\mid$	$\emptyset$	empty dirt
	$\mid$	$\Delta_1 \sqcap \Delta_2$	intersection
	$\mid$	$\Delta_1 \sqcup \Delta_2$	union
All operations Ω	$::=$	$\{\mathtt{Op}_i \mid \mathtt{Op}_i \in \Sigma\}$	

Figuur 5.2: Types of EffCore

$$A_1 \leqslant A_2 \leftrightarrow A_1 \sqcup A_2 \equiv A_2$$

$$A_1 \leqslant A_2 \leftrightarrow A_1 \equiv A_1 \sqcap A_2$$

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

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

$$\underline{C}_1 \leqslant \underline{C}_2 \leftrightarrow \underline{C}_1 \sqcup \underline{C}_2 \equiv \underline{C}_2$$

$$\underline{C}_1 \leqslant \underline{C}_2 \leftrightarrow \underline{C}_1 \equiv \underline{C}_1 \sqcap \underline{C}_2$$

Figuur 5.3: Relationship between Equivalence and Subtyping

$$A \sqcup A \equiv A \qquad A \sqcap A \equiv A$$

$$A_1 \sqcup A_2 \equiv A_2 \sqcup A_1 \qquad A_1 \sqcap A_2 \equiv A_2 \sqcap A_1$$

$$A_1 \sqcup (A_2 \sqcup A_3) \equiv (A_1 \sqcup A_2) \sqcup A_3 \qquad A_1 \sqcap (A_2 \sqcap A_3) \equiv (A_1 \sqcap A_2) \sqcap A_3$$

$$A_1 \sqcup (A_1 \sqcap A_2) \equiv A_1 \qquad A_1 \sqcap (A_1 \sqcup A_2) \equiv A_1$$

$$\bot \sqcup A \equiv A \qquad \bot \sqcap A \equiv \bot$$

$$\top \sqcup A \equiv \top \qquad \top \sqcap A \equiv A$$

$$A_1 \sqcup (A_2 \sqcap A_3) \equiv (A_1 \sqcup A_2) \sqcap (A_1 \sqcup A_3)$$

$$A_1 \sqcap (A_2 \sqcup A_3) \equiv (A_1 \sqcap A_2) \sqcup (A_1 \sqcap A_3)$$

Figuur 5.4: Equations of distributive lattices for types

5.2 Type system

5.3 Typing rules

Figure 5.8 defines the typing judgements for values and computations with respect to a standard typing context Γ.

$$(A_1 \to A_2) \sqcup (A_3 \to A_4) \equiv (A_1 \sqcap A_3) \to (A_2 \sqcup A_4)$$

$$(A_1 \to A_2) \sqcap (A_3 \to A_4) \equiv (A_1 \sqcup A_3) \to (A_2 \sqcap A_4)$$

$$(A_1 \Rightarrow A_2) \sqcup (A_3 \Rightarrow A_4) \equiv (A_1 \sqcap A_3) \Rightarrow (A_2 \sqcup A_4)$$

$$(A_1 \Rightarrow A_2) \sqcap (A_3 \Rightarrow A_4) \equiv (A_1 \sqcup A_3) \Rightarrow (A_2 \sqcap A_4)$$

$$(\underline{C}_1 \sqcup \underline{C}_2) \equiv (A_1 \mathbin{!} \Delta_1 \sqcup A_2 \mathbin{!} \Delta_2) \equiv (A_1 \sqcup A_2) \mathbin{!} (\Delta_1 \sqcup \Delta_2)$$

$$(\underline{C}_1 \sqcap \underline{C}_2) \equiv (A_1 \mathbin{!} \Delta_1 \sqcap A_2 \mathbin{!} \Delta_2) \equiv (A_1 \sqcap A_2) \mathbin{!} (\Delta_1 \sqcap \Delta_2)$$

Figuur 5.5: Equations for function, handler and dirty types

$$\Delta \sqcup \Delta \equiv \Delta \qquad \Delta \sqcap \Delta \equiv \Delta$$

$$\Delta_1 \sqcup \Delta_2 \equiv \Delta_2 \sqcup \Delta_1 \qquad \Delta_1 \sqcap \Delta_2 \equiv \Delta_2 \sqcap \Delta_1$$

$$\Delta_1 \sqcup (\Delta_2 \sqcup \Delta_3) \equiv (\Delta_1 \sqcup \Delta_2) \sqcup \Delta_3 \qquad \Delta_1 \sqcap (\Delta_2 \sqcap \Delta_3) \equiv (\Delta_1 \sqcap \Delta_2) \sqcap \Delta_3$$

$$\Delta_1 \sqcup (\Delta_1 \sqcap \Delta_2) \equiv \Delta_1 \qquad \Delta_1 \sqcap (\Delta_1 \sqcup \Delta_2) \equiv \Delta_1$$

$$\emptyset \sqcup \Delta \equiv \Delta \qquad \emptyset \sqcap \Delta \equiv \emptyset$$

$$\Omega \sqcup \Delta \equiv \Omega \qquad \Omega \sqcap \Delta \equiv \Delta$$

$$\Delta_1 \sqcup (\Delta_2 \sqcap \Delta_3) \equiv (\Delta_1 \sqcup \Delta_2) \sqcap (\Delta_1 \sqcup \Delta_3)$$

$$\Delta_1 \sqcap (\Delta_2 \sqcup \Delta_3) \equiv (\Delta_1 \sqcap \Delta_2) \sqcup (\Delta_1 \sqcap \Delta_3)$$

Figuur 5.6: Equations of distributive lattices for dirts

Values

The rules for subtyping, variables, type abstraction, type application and functions are entirely standard. For constants we assume a signature Σ that assigns a type A to each constant `k`, which we write as $(\mathtt{k} : A) \in \Sigma$.

A handler expression has type $A \mathbin{!} \Delta \cup \mathcal{O} \Rightarrow B \mathbin{!} \Delta$ iff all branches (both the operation cases and the return case) have dirty type $B \mathbin{!} \Delta$ and the operation cases cover the set of

Subtyping of dirts

Sub-!-Row-Row
$$\frac{n \geq 0 \qquad m \geq 0 \qquad p \geq 0 \qquad \{Op_1, ..., Op_n, Op_{n+m+1}, ..., Op_{n+m+p}, \delta_1\} \leqslant \{Op_1, ..., Op_n, Op_{n+1}, ..., Op_{n+m}, \delta_2\}}{\{\delta_1\} \leqslant \{Op_{n+1}, ..., Op_{n+m}, \delta_3\} \qquad \{\delta_3\} = \{Op_{n+m}, ..., Op_{n+m+p}, \delta_2\}}$$

Sub-!-Dot-Row
$$\frac{n \geq 0 \qquad m \geq 0 \qquad p \geq 0 \qquad \{Op_1, ..., Op_n, Op_{n+m+1}, ..., Op_{n+m+p}, .\} \leqslant \{Op_1, ..., Op_n, Op_{n+1}, ..., Op_{n+m}, \delta_2\}}{\emptyset \leqslant \{Op_{n+1}, ..., Op_{n+m}, \delta_3\} \qquad \{\delta_3\} = \{Op_{n+m}, ..., Op_{n+m+p}, \delta_2\}}$$

Sub-!-Row-Dot
$$\frac{n \geq 0 \qquad m \geq 0 \qquad \{Op_1, ..., Op_n, \delta_1\} \leqslant \{Op_1, ..., Op_n, Op_{n+1}, Op_{n+m}, .\}}{\{\delta_1\} \leqslant \{Op_{n+1}, Op_{n+m}, .\}}$$

Sub-!-Dot-Dot
$$\frac{n \geq 0 \qquad m \geq 0 \qquad \{Op_1, ..., Op_n, .\} \leqslant \{Op_1, ..., Op_n, Op_{n+1}, ..., Op_{n+m}, .\}}{\emptyset \leqslant \{Op_{n+1}, Op_{n+m}, .\}}$$

Figuur 5.7: Subtyping for dirts of EffCore

operations O. Note that the intersection $\Delta \cap O$ is not necessarily empty (with $\cap$ being the intersection of the operations, not to be confused with the $\sqcap$ type). The handler deals with the operations O, but in the process may re-issue some of them (i.e., $\Delta \cap O$).

When typing operation cases, the given signature for the operation $(\mathtt{Op} : A_{\mathtt{Op}} \to B_{\mathtt{Op}}) \in \Sigma$ determines the type $A_{\mathtt{Op}}$ of the parameter x and the domain $B_{\mathtt{Op}}$ of the continuation k. As our handlers are deep, the codomain of k should be the same as the type $B \mathrel{!} \Delta$ of the cases.

Computations

With the following exceptions, the typing judgement $\Gamma \vdash c : \underline{C}$ has a straightforward definition. The `return` construct renders a value v as a pure computation, i.e., with empty dirt. In this case, this is defined as a set with the .(DOT) as the only element. An operation invocation $\mathtt{Op}\, v$ is typed according to the operation's signature, with the operation itself as its only operation. Finally, rule With shows that a handler with type $\underline{C} \Rightarrow \underline{D}$ transforms a computation with type $\underline{C}$ into a computation with type $\underline{D}$.

5.4 Reformulated typing rules

5.5 Semantics

typing contexts $\Gamma \; ::= \; \epsilon \mid \Gamma, x : A \mid \Gamma, \hat{\mathtt{x}} : \forall \bar{\alpha}.B$

Expressions

$$\textsc{Sub-Val}\quad \frac{\Gamma \vdash v : A \qquad A \leqslant B}{\Gamma \vdash v : B} \qquad \textsc{Var-}\lambda\quad \frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{Var-}\forall\quad \frac{(\hat{\mathtt{x}} : \forall \bar{\alpha}.A) \in \Gamma}{\Gamma \vdash \hat{\mathtt{x}} : A[\bar{A}/\bar{\alpha}]} \qquad \textsc{True}\quad \frac{}{\Gamma \vdash \mathtt{true} : \mathit{bool}}$$

$$\textsc{False}\quad \frac{}{\Gamma \vdash \mathtt{false} : \mathit{bool}} \qquad \textsc{Fun}\quad \frac{\Gamma, x : A \vdash c : \underline{C}}{\Gamma \vdash \lambda x.c : A \to \underline{C}}$$

$$\textsc{Hand}\quad \frac{\begin{matrix} & \Gamma, x : A \vdash c_r : B \,!\, \Delta \\ \Big[(\mathtt{Op} : A_{\mathtt{Op}} \to B_{\mathtt{Op}}) \in \Sigma & \Gamma, x : A_{\mathtt{Op}}, k : B_{\mathtt{Op}} \to B \,!\, \Delta \vdash c_{\mathtt{Op}} : B \,!\, \Delta\Big]_{\mathtt{Op} \in O} \end{matrix}}{\Gamma \vdash \{\mathtt{return}\ x \mapsto c_r, [\mathtt{Op}\ y\ k \mapsto c_{\mathtt{Op}}]_{\mathtt{Op} \in O}\} : \quad A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta}$$

Computations

$$\textsc{Sub-Comp}\quad \frac{\Gamma \vdash c : \underline{C} \qquad \underline{C} \leqslant \underline{D}}{\Gamma \vdash c : \underline{D}} \qquad \textsc{App}\quad \frac{\Gamma \vdash v_1 : A \to \underline{C} \qquad \Gamma \vdash v_2 : A}{\Gamma \vdash v_1\, v_2 : \underline{C}}$$

$$\textsc{Cond}\quad \frac{\Gamma \vdash v : \mathit{bool} \qquad \Gamma \vdash c_1 : \underline{C} \qquad \Gamma \vdash c_2 : \underline{C}}{\Gamma \vdash \mathtt{if}\ v\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 : \underline{C}} \qquad \textsc{Ret}\quad \frac{\Gamma \vdash v : A}{\Gamma \vdash \mathtt{return}\ v : A \,!\, \emptyset}$$

$$\textsc{Op}\quad \frac{(\mathtt{Op} : A \to B) \in \Sigma \qquad \Gamma \vdash v : A \qquad \underline{C} : B \,!\, \{\mathtt{Op}\ ;\ R\}}{\Gamma \vdash \mathtt{Op}\, v : \underline{C}}$$

$$\textsc{Let}\quad \frac{\Gamma \vdash c_1 : A \,!\, \Delta \qquad \Gamma, x : \forall \bar{\alpha}.A \vdash c_2 : B \,!\, \Delta \qquad \alpha \notin \mathit{FTV}(\Gamma)}{\Gamma \vdash \mathtt{let}\ \hat{\mathtt{x}} = c_1\ \mathtt{in}\ c_2 : B \,!\, \Delta}$$

$$\textsc{With}\quad \frac{\Gamma \vdash v : \underline{C} \Rightarrow \underline{D} \qquad \Gamma \vdash c : \underline{C}}{\Gamma \vdash \mathtt{handle}\ c\ \mathtt{with}\ v : \underline{D}}$$

Figuur 5.8: Typing of EffCore

$$\begin{array}{rcl} \text{monomorphic typing contexts } \Xi & ::= & \epsilon \mid \Xi, x : A \\ \text{polymorphic typing contexts } \Pi & ::= & \epsilon \mid \Pi, \hat{\mathtt{x}} : [\Xi]A \mid \Pi, \hat{\mathtt{x}} : [\Xi]\underline{C} \end{array}$$

Expressions

$$\textsc{Sub-Val}\ \frac{\Pi \Vdash v : [\Xi_1]A_1 \qquad [\Xi_1]A_1 \leqslant^{\forall} [\Xi_2]A_2}{\Pi \Vdash v : [\Xi_2]A_2} \qquad \textsc{Var-}\lambda\ \frac{}{\Pi \Vdash x : [x : A]A} \qquad \textsc{Var-}\forall\ \frac{(\hat{\mathtt{x}} : [\Xi]A) \in \Pi}{\Pi \Vdash \hat{\mathtt{x}} : [\Xi]A}$$

$$\textsc{True}\ \frac{}{\Pi \Vdash \mathtt{true} : []\mathit{bool}} \qquad \textsc{False}\ \frac{}{\Pi \Vdash \mathtt{false} : []\mathit{bool}} \qquad \textsc{Fun}\ \frac{\Pi \Vdash c : [\Xi, x : A]\underline{C}}{\Pi \Vdash \lambda x.c : [\Xi]A \to \underline{C}}$$

$$\textsc{Hand}\ \frac{\begin{array}{c} \Gamma, x : A \vdash c_r : B \,!\, \Delta \\ \left[(\mathtt{Op} : A_{\mathtt{Op}} \to B_{\mathtt{Op}}) \in \Sigma \qquad \Gamma, x : A_{\mathtt{Op}}, k : B_{\mathtt{Op}} \to B \,!\, \Delta \vdash c_{\mathtt{Op}} : B \,!\, \Delta\right]_{\mathtt{Op} \in O} \end{array}}{\Gamma \vdash \{\mathtt{return}\ x \mapsto c_r, [\mathtt{Op}\ y\ k \mapsto c_{\mathtt{Op}}]_{\mathtt{Op} \in O}\} : \quad A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta}$$

Computations

$$\textsc{Sub-Comp}\ \frac{\Pi \Vdash c : [\Xi_1]\underline{C}_1 \qquad [\Xi_1]\underline{C}_1 \leqslant^{\forall} [\Xi_2]\underline{C}_2}{\Pi \Vdash c : [\Xi_2]\underline{C}_2} \qquad \textsc{App}\ \frac{\Pi \Vdash v_1 : [\Xi]A \to \underline{C} \qquad \Pi \Vdash v_2 : [\Xi]A}{\Pi \Vdash v_1\, v_2 : [\Xi]\underline{C}}$$

$$\textsc{Cond}\ \frac{\Pi \Vdash v : [\Xi]\mathit{bool} \qquad \Pi \Vdash c_1 : [\Xi]\underline{C} \qquad \Pi \Vdash c_2 : [\Xi]\underline{C}}{\Pi \Vdash \mathtt{if}\ v\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 : [\Xi](\underline{C})} \qquad \textsc{Ret}\ \frac{\Pi \Vdash v : [\Xi]A}{\Pi \Vdash \mathtt{return}\ v : [\Xi]A \,!\, \emptyset}$$

$$\textsc{Op}\ \frac{(\mathtt{Op} : A \to B) \in \Sigma \qquad \Pi \Vdash v : [\Xi]A \qquad \underline{C} : [\Xi]B \,!\, \{\mathtt{Op}\ ;\ R\}}{\Pi \Vdash \mathtt{Op}\, v : [\Xi]\underline{C}}$$

$$\textsc{Let}\ \frac{\Pi \Vdash c_1 : [\Xi_1]A \,!\, \Delta \qquad \Pi, x : [\Xi_1]A \Vdash c_2 : [\Xi_2]B \,!\, \Delta}{\Gamma \vdash \mathtt{let}\ \hat{\mathtt{x}} = c_1\ \mathtt{in}\ c_2 : [\Xi_1 \sqcap \Xi_2]B \,!\, \Delta}$$

$$\textsc{With}\ \frac{\Pi \Vdash v : [\Xi]\underline{C} \Rightarrow \underline{D} \qquad \Pi \Vdash c : [\Xi]\underline{C}}{\Pi \Vdash \mathtt{handle}\ c\ \mathtt{with}\ v : [\Xi]\underline{D}}$$

Figuur 5.9: Reformulated typing rules of EffCore

Hoofdstuk 6

Type Inference

6.1 Elaboration of Eff into EffCore

The elaboration show how the source language can be transformed into EffCore.

$$\text{typing contexts } \Gamma \ ::= \ \epsilon \ \mid \ \Gamma, x : A \ \mid \ \Gamma, x : \forall \bar{\alpha}.B$$

Expressions

$$\textsc{Val}\ \frac{\Gamma \vdash v : A \rightsquigarrow v' \qquad \vdash_c A \leqslant B}{\Gamma \vdash v : B \rightsquigarrow v'} \qquad \textsc{Var}\ \frac{(x : S) \in \Gamma \qquad S = \forall \bar{\alpha}.A}{\Gamma \vdash x : A[\bar{S}/\bar{\alpha}] \rightsquigarrow x\,\bar{S}} \qquad \textsc{Const}\ \frac{(\mathtt{k} : A) \in \Sigma}{\Gamma \vdash \mathtt{k} : A \rightsquigarrow \mathtt{k}'}$$

$$\textsc{Fun}\ \frac{\Gamma, x : A \vdash c : \underline{C} \rightsquigarrow c'}{\Gamma \vdash \lambda x.c : A \to \underline{C} \rightsquigarrow \mathtt{fun}\ x : A \mapsto c' : A \to \underline{C}}$$

$$\textsc{Hand}\ \frac{\begin{array}{c} \Gamma, x : A \vdash c_r : B \,!\, \Delta \\ \left[(\mathtt{Op} : A_{\mathtt{Op}} \to B_{\mathtt{Op}}) \in \Sigma \qquad \Gamma, x : A_{\mathtt{Op}}, k : B_{\mathtt{Op}} \to B \,!\, \Delta \vdash c_{\mathtt{Op}} : B \,!\, \Delta \right]_{\mathtt{Op} \in O} \end{array}}{\begin{array}{ll} \Gamma \vdash \{\mathtt{return}\ x \mapsto c_r, [\mathtt{Op}\ y\ k \mapsto c_{\mathtt{Op}}]_{\mathtt{Op} \in O}\} : & A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta \\ \rightsquigarrow \{\mathtt{return}\ x \mapsto c_r, [\mathtt{Op}\ y\ k \mapsto c_{\mathtt{Op}}]_{\mathtt{Op} \in O}\} : & A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta \end{array}}$$

Figuur 6.1: Elaboration of source to core language: expressions

6.2 Constraint Generation

Figure 6.3 and Figure 6.4 show the constraint generation algorithm for EffCore.

typing contexts $\Gamma \;::=\; \epsilon \mid \Gamma, x : A, x : \forall\bar{\alpha}.B$

Computations

Comp
$$\frac{\Gamma \vdash c : \underline{C} \rightsquigarrow c' \qquad \underline{C} \leqslant \underline{D}}{\Gamma \vdash c : \underline{D} \rightsquigarrow c'}$$

App
$$\frac{\Gamma \vdash v_1 : A \to \underline{C} \rightsquigarrow v_1' \qquad \Gamma \vdash v_2 : A \rightsquigarrow v_2'}{\Gamma \vdash v_1\, v_2 : \underline{C} \rightsquigarrow v_1'\, v_2' : \underline{C}}$$

Ret
$$\frac{\Gamma \vdash v : A \rightsquigarrow v'}{\Gamma \vdash \mathtt{return}\; v : A \,!\, \emptyset \rightsquigarrow \mathtt{return}\; v' : A \,!\, \emptyset}$$

Op
$$\frac{(\mathtt{Op} : A \to B) \in \Sigma \qquad \Gamma \vdash v : A \rightsquigarrow v'}{\Gamma \vdash \mathtt{Op}\, v : B \,!\, \{\mathtt{Op}\} \rightsquigarrow \mathtt{Op}\, v' : B \,!\, \{\mathtt{Op}, .\}}$$

Do
$$\frac{\begin{array}{c}\Gamma \vdash c_1 : \underline{C} \rightsquigarrow c_1' \\ S = \forall\bar{\alpha}.A \qquad \bar{\alpha} = FTV(A) - TV(\Gamma) \qquad \Gamma, x : S \vdash c_2 : \underline{D} \rightsquigarrow c_2'\end{array}}{\Gamma \vdash \mathtt{do}\; x \leftarrow c_1 \,;\, c_2 : \underline{D} \rightsquigarrow (\mathtt{fun}\; x : A \mapsto c_2')(\Lambda\bar{\alpha}.c_1')}$$

With
$$\frac{\Gamma \vdash v : \underline{C} \Rightarrow \underline{D} \rightsquigarrow v' \qquad \Gamma \vdash c : \underline{C} \rightsquigarrow c'}{\Gamma \vdash \mathtt{handle}\; c\; \mathtt{with}\; v : \underline{D} \rightsquigarrow \mathtt{handle}\; c'\; \mathtt{with}\; v' : \underline{D}}$$

Figuur 6.2: Elaboration of source to core language: computations

6.3 Polar types

6.4 Unification

typing contexts $\Gamma \ ::= \ \ \epsilon \ \mid \ \Gamma, x : A, x : \forall\bar{\alpha}.B$

Expressions

$$\textsc{Val}\ \frac{\Gamma \vdash v : A \rightsquigarrow v' \qquad A \leqslant B}{\Gamma \vdash v : B \rightsquigarrow v'} \qquad \textsc{Var}\ \frac{(x : S) \in \Gamma \qquad S = \forall\bar{\alpha}.A}{\Gamma \vdash x : A[\bar{S}/\bar{\alpha}] \rightsquigarrow x\,\bar{S}} \qquad \textsc{Const}\ \frac{(\mathtt{k} : A) \in \Sigma}{\Gamma \vdash \mathtt{k} : A \rightsquigarrow \mathtt{k}'}$$

$$\textsc{Fun}\ \frac{\Gamma, x : A \vdash c : \underline{C} \rightsquigarrow c'}{\Gamma \vdash \lambda x.c : A \to \underline{C} \rightsquigarrow \mathtt{fun}\ x : A \mapsto c' : A \to \underline{C}}$$

$$\textsc{Hand}\ \frac{\begin{array}{c}\Gamma, x : A \vdash c_r : B \,!\, \Delta \\ \Big[(\mathtt{Op} : A_{\mathtt{Op}} \to B_{\mathtt{Op}}) \in \Sigma \qquad \Gamma, x : A_{\mathtt{Op}}, k : B_{\mathtt{Op}} \to B \,!\, \Delta \vdash c_{\mathtt{Op}} : B \,!\, \Delta\Big]_{\mathtt{Op} \in O}\end{array}}{\begin{array}{l}\Gamma \vdash \{\mathtt{return}\ x \mapsto c_r, [\mathtt{Op}\ y\ k \mapsto c_{\mathtt{Op}}]_{\mathtt{Op} \in O}\} : \quad A \,!\, \Delta \cup O \Rrightarrow B \,!\, \Delta \\ \rightsquigarrow \{\mathtt{return}\ x \mapsto c_r, [\mathtt{Op}\ y\ k \mapsto c_{\mathtt{Op}}]_{\mathtt{Op} \in O}\} : \quad A \,!\, \Delta \cup O \Rrightarrow B \,!\, \Delta\end{array}}$$

Figuur 6.3: Constraint generation within expressions

$$\text{typing contexts } \Gamma \;::=\; \epsilon \;\mid\; \Gamma, x : A, x : \forall\bar{\alpha}.B$$

Computations

$$\textsc{Comp}\quad \frac{\Gamma \vdash c : \underline{C} \rightsquigarrow c' \qquad \underline{C} \leqslant \underline{D}}{\Gamma \vdash c : \underline{D} \rightsquigarrow c'}
\qquad\qquad
\textsc{App}\quad \frac{\Gamma \vdash v_1 : A \to \underline{C} \rightsquigarrow v_1' \qquad \Gamma \vdash v_2 : A \rightsquigarrow v_2'}{\Gamma \vdash v_1\, v_2 : \underline{C} \rightsquigarrow v_1'\, v_2' : \underline{C}}$$

$$\textsc{LetRec}\quad \frac{\Gamma, f : A \to \underline{C}, x : A \vdash c_1 : \underline{C} \rightsquigarrow c_1' s \qquad \Gamma, f : A \to \underline{C} \vdash c_2 : \underline{D} \rightsquigarrow c_2'}{\Gamma \vdash \mathtt{let\ rec}\ f\ x = c_1\ \mathtt{in}\ c_2 : \underline{D} \qquad \rightsquigarrow \mathtt{let\ rec}\ f\ x = c_1'\ \mathtt{in}\ c_2' : \underline{D}}$$

$$\textsc{Ret}\quad \frac{\Gamma \vdash v : A \rightsquigarrow v'}{\Gamma \vdash \mathtt{return}\ v : A \,!\, \emptyset \rightsquigarrow \mathtt{return}\ v' : A \,!\, \emptyset}$$

$$\textsc{Op}\quad \frac{(\mathtt{Op} : A \to B) \in \Sigma \qquad \Gamma \vdash v : A \rightsquigarrow v'}{\Gamma \vdash \mathtt{Op}\, v : B \,!\, \{\mathtt{Op}\} \rightsquigarrow \mathtt{Op}\, v' : B \,!\, \{\mathtt{Op}, .\}}$$

$$\textsc{Do}\quad \frac{\begin{array}{c}\Gamma \vdash c_1 : \underline{C} \rightsquigarrow c_1' \\ S = \forall\bar{\alpha}.A \qquad \bar{\alpha} = FTV(A) - TV(\Gamma) \qquad \Gamma, x : S \vdash c_2 : \underline{D} \rightsquigarrow c_2'\end{array}}{\Gamma \vdash \mathrm{do}\ x \leftarrow c_1\ ;\ c_2 : \underline{D} \rightsquigarrow (\mathtt{fun}\ x : A \mapsto c_2')(\Lambda\bar{\alpha}.c_1')}$$

$$\textsc{With}\quad \frac{\Gamma \vdash v : \underline{C} \Rightarrow \underline{D} \rightsquigarrow v' \qquad \Gamma \vdash c : \underline{C} \rightsquigarrow c'}{\Gamma \vdash \mathtt{handle}\ c\ \mathtt{with}\ v : \underline{D} \rightsquigarrow \mathtt{handle}\ c'\ \mathtt{with}\ v' : \underline{D}}$$

Figuur 6.4: Constraint generation within computations

(pure) type A^+, B^+	$::=$	`bool`	bool type
	$\mid$	$A^- \rightarrow \underline{C}^+$	function type
	$\mid$	$\underline{C}^- \Rightarrow \underline{D}^+$	**handler type**
	$\mid$	α	type variable
	$\mid$	$\mu\alpha.A^+$	recursive type
	$\mid$	$\bot$	bottom
	$\mid$	$A^+ \sqcup B^+$	union
dirty type $\underline{C}^+, \underline{D}^+$	$::=$	$A^+ \,!\, \Delta^+$	
(pure) type A^-, B^-	$::=$	`bool`	bool type
	$\mid$	$A^+ \rightarrow \underline{C}^-$	function type
	$\mid$	$\underline{C}^+ \Rightarrow \underline{D}^-$	**handler type**
	$\mid$	α	type variable
	$\mid$	$\mu\alpha.A^-$	recursive type
	$\mid$	$\top$	top
	$\mid$	$A^- \sqcap B^-$	intersection
dirty type $\underline{C}^-, \underline{D}^-$	$::=$	$A^- \,!\, \Delta^-$	
dirt Δ	$::=$	`Op`	operation
	$\mid$	δ	row variable
	$\mid$	$\emptyset$	empty dirt
	$\mid$	$\Delta_1 \sqcap \Delta_2$	intersection
	$\mid$	$\Delta_1 \sqcup \Delta_2$	union
All operations Ω	$::=$	$\{\mathtt{Op}_i \mid \mathtt{Op}_i \in \Sigma\}$	

Figuur 6.5: Polar types of EffCore

Hoofdstuk 7

Proofs

Hoofdstuk 8

Implementation

Describe the approach itself, in such detail that a reader could also implement this approach if s/he wished to do that.

Hoofdstuk 9

Evaluation

Novel approaches to problems are often evaluated empirically. Describe the evaluation process in such detail that a reader could reproduce the results. Describe in detail the setup of an experiment. Argue why this experiment is useful, and what you could learn from it. Be precise about what you want to measure, or about the hypothesis that you are testing. Discuss and interpret the results in terms of your experimental questions. Summarize the conclusions of the experimental evaluation.

Hoofdstuk 10

Conclusion

Briefly recall what the goal of the work was. Summarize what you have done, summarize the results, and present conclusions. Conclusions include a critical assessment: where the original goals reached? Discuss the limitations of your work. Describe how the work could possibly be extended in the future, mitigating limitations or solving remaining problems.

Bijlagen

Bibliografie

[1] A. Bauer and M. Pretnar. An effect system for algebraic effects and handlers. *Logical Methods in Computer Science*, 10(4), 2014.

Fiche masterproef

Student: Axel Faes

Titel: Algebraic Subtyping for Algebraic Types and Effects

UDC:

Korte inhoud:

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotor: Prof. dr. ir. Tom Schrijvers

Assessor: assesors

Begeleider: Amr Hany Saleh