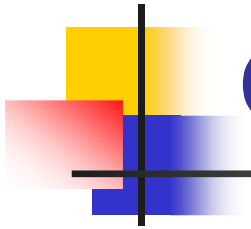
Subtyping and Computational Monads

Bachelor Thesis

Introduction Talk

Dieter Brunotte

Advisor: Jan Schwinghammer



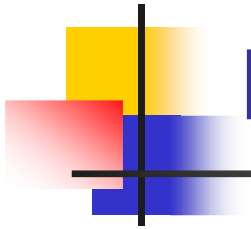
Computational Monads

- Programming languages have effects (exceptions, I/O-operations, divergence)

- „pure“ lambda calculus add effects in ad-hoc manner

-> have new typing/reduction rules

-> basic properties (e.g. Preservation, Progress) must be proved again



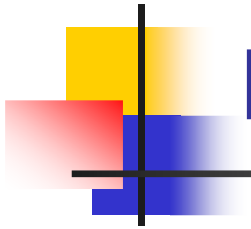
Example: I/O-operations

- „pure“ lambda-calculus :
  t ::= x| λx:T.t | tt

- With I/O:
  s ϵ ∑* (output alphabet)
  t ::= …| print s

- New typing rules:

$$\frac{}{\Gamma \vdash \text{print } s : \text{unit}}$$



Example: I/O-operations (2)

- New evaluation rules: (add outputstring to terms)

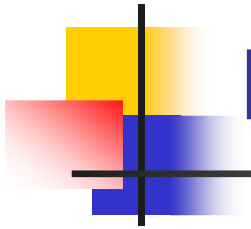
$$s \mid t \rightarrow s \mid t' \quad \text{where } t \rightarrow t' \text{ in pure calculus}$$

$$s \mid \text{print } s' \rightarrow s \bullet s' \mid ( )$$

- Example : $(\lambda x : \text{unit. } x;\ x)\ (\text{print } s)$
- Nondeterministic evaluation with different effects

$$\ldots \rightarrow s \bullet s \mid ( )$$

$$\ldots \rightarrow s \mid ( )$$



Example 2: side-effects

- New type constructor ref with new typing rules
- New constants/operations

$$\mathrm{read}_A : A\,\mathrm{ref} \to A$$

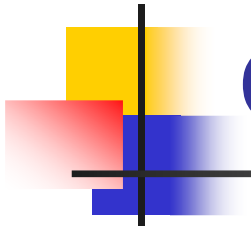
$$\mathrm{write}_A : A\,\mathrm{ref} \to A \to \mathrm{unit}$$

- For evaluation add starage:

$$S \mid t \to S \mid t' \quad \text{where } t \to t' \text{ in pure calculus}$$

$$S \mid \mathrm{read}_A\ l \to S \mid S\,l$$

$$S \mid \mathrm{write}_A\ l\,e \to S[l := e] \mid ()$$



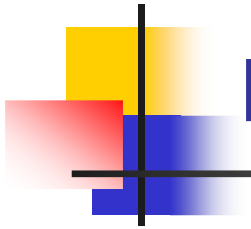
Computational Monads

- Monadic types distinguishes between pure terms and effectful computations
- Unrestricted β/η-reductions
- Effect synchronisation by let-construction

Moggis calculus

$$A, B \in \text{Types} ::= \text{Top} \mid b \mid A \to B \mid \{l_i : A_i^{\,i \in 1 \ldots n}\}$$
$$\mid T\ A$$

$$t \in \text{Ter} ::= \ldots \mid \{l_i = t_i^{\,i=1 \ldots n}\} \mid t.l \mid$$
$$\mid [t] \mid \text{let } x \Leftarrow t \text{ in } t$$



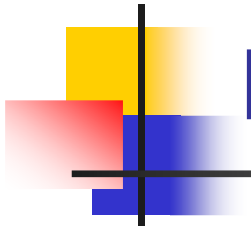
New typing rules for monads

for new monadic unit :

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash [t] : T\ A}$$

for let - expression :

$$\frac{\Gamma \vdash t' : T\ A \qquad \Gamma, x : A \vdash t : T\ B}{\Gamma \vdash \text{let } x \Leftarrow t' \text{ in } t : T\ B}$$



Reduction

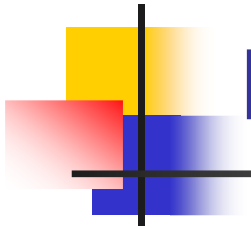
$$(\beta.\text{let}) \quad \text{let } x \Leftarrow [\,e'\,] \text{ in } e \;\rightarrow\; e[x := e']$$

- Simulates application (λx.e) e‘ for monads
- requires full evaluation of argument to monadic value

$$(\eta.\text{let}) \quad \text{let } x \Leftarrow e \text{ in } [x] \;\rightarrow\; e$$

$$(\text{assoc}) \quad \text{let } y \Leftarrow (\text{let } x \Leftarrow e' \text{ in } e'') \text{ in } e$$

$$\rightarrow \text{let } x \Leftarrow e' \text{ in } (\text{let } y \Leftarrow e'' \text{ in } e)$$



Example 1: I/O with Monads

- Intuitively T A = Σ^* x A
- [e] ~ outputs ε, returns e
- Let x <= e‘ in e

~ Concatenates all output of e‘ with output of „e[x:=e‘]“

- Print s ~ output s, return [{}]

Example 1: I/O with Monads

- effects on reduction rules

$$\text{print}\, s \xrightarrow{s} [\{\}]$$

$$\frac{e \to e'}{C[e] \xrightarrow{\varepsilon} C[e']} \quad \text{for arbitrary context } C$$

$$\frac{e \xrightarrow{s} e'}{D[e] \xrightarrow{s} D[e']} \quad \text{for } D ::= \circ \mid \text{let } x \Leftarrow D \text{ in } t$$

Example 1: I/O with Monads

- Top level reduction of monadic calculus

$$\frac{e \xrightarrow{s'} e'}{s \mid e \;\rightarrow s \bullet s' \mid e'}$$

Example 2:side-effects

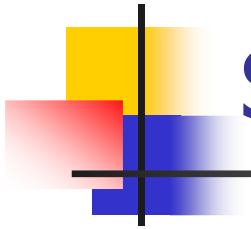
$$(\beta.\text{read}) \quad \text{read}\, l \xrightarrow{?l=e} [e]$$

$$(\beta.\text{write}) \quad \text{write}\, l\, e \xrightarrow{l:=e} [\{\}]$$

- Observe Ø otherwise

- Top level reduction of monadic calculus

$$\frac{e \xrightarrow{?l=t} e'}{S \mid e \rightarrow S \mid e'} \text{ if } S(l) = t \qquad \frac{e \xrightarrow{l:=t} e'}{S \mid e \rightarrow S[l := t] \mid e'}$$



Subtyping

- Allow to use term of type A where type B is expected iff A <: B (A subtype of B)

- Subtype rules:

  example functions (contravariance)

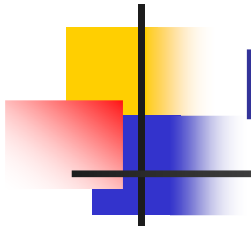
$$\frac{B_1 <: A_1 \quad A_2 <: B_2}{A_1 \to A_2 <: B_1 \to B_2}$$

Subtyping and Monadic Types

- Which subtype rule holds for monad:

$$\frac{A <: B}{T\ A <: T\ B}\ ? \qquad \frac{A <: B \quad B <: A}{T\ A <: T\ B}\ ?$$

- Soundness proof of Moggis calculus with Subtyping (Preservation,Progress)



References

- Eugenio Moggi. Notions of computation and monads. Information and Computation, 93,55-92, 1991.
- Nick Benton & John Hughes & Eugenio Moggi. Monads and effects. Pages 42-122 of Advanced lectures from int. summer school on applied semantics. LNCS vol. 2395, Springer, 2002.
- Benjamin Pierce. Types and Programming Languages. MIT Press, 2002.
- Philip Wadler and Peter Thiemann. The marriage of effects and monads .ACM Transactions on Computational Logic,1(4),1-32,2003.